AFIT/GSO/ENY/99M-09

ANTI-BALLISTIC MISSILE LASER PREDICTIVE AVOIDANCE OF SATELLITES: THEORY AND SOFTWARE FOR REAL-TIME PROCESSING AND DECONFLICTION OF SATELLITE EPHEMERIDES WITH A MOVING PLATFORM LASER

THESIS
(Book 2 of 2)

David J. Vloedman, Captain, USAF

AFIT/GSO/ENY/99M-09

Appendix D.
Software Library Implementation Code

D.1 Aircraft.cpp

```
/**************************************************************************/
/*  MODULE NAME:      Aircraft.cpp                                        */
/*  AUTHOR:           Captain David Vloedman                              */
/*  DATE CREATED:     Sept 20, 1998                                       */
/*                                                                        */
/*  PURPOSE:          This module of code houses the Aircraft class object. */
/*                                                                        */
/*  COMPILER:         Borland C++ Builder3 Standard version               */
/*                    This compiler should be used to compile and link.   */
/*                                                                        */
/**************************************************************************/
/*********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/*********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/*********************************/
/* USER-BUILT LIBRARIES          */
/*********************************/
#include "Aircraft.h"
/*********************************/
/* C GENERAL LIBRARIES           */
/*********************************/
#include <stdio.h>
#include <iostream.h>

/****************************************/
/*   CREATE THE AIRCRAFT CONSTRUCTOR    */
/****************************************/
Aircraft::Aircraft() :
    LatitudeDegree(0),
    LatitudeMinute(0),
    LatitudeSecond(0),
    LatitudeHemisphere(0),
    LongitudeDegree(0),
    LongitudeMinute(0),
    LongitudeSecond(0),
    VelocityX(0),
    VelocityY(0),
    VelocityZ(0),
    Altitude(0)
    {

    }

/****************************************/
/*   CREATE THE AIRCRAFT DESTRUCTOR     */
/****************************************/

Aircraft::~Aircraft()
    {
    }

/**************************************************************************/
```

```
/********************   AIRCRAFT MANIPULATION FUNCTIONS   ******************/
/**************************************************************************/

/*****************************************/
/*   SET LATITUDE DEGREE                 */
/*****************************************/
void Aircraft::SetLatitudeDegree(int ld)
{    LatitudeDegree = ld; }

/*****************************************/
/*   SET LATITUDE MINUTE                 */
/*****************************************/
void Aircraft::SetLatitudeMinute(int lm)
{  LatitudeMinute = lm; }

/*****************************************/
/*   SET LATITUDE SECOND                 */
/*****************************************/
void Aircraft::SetLatitudeSecond(double ls)
{   LatitudeSecond = ls;  }

/*****************************************/
/*   SET LATITUDE HEMIPHERE              */
/*  LatitudeHemisphere = "0" = NORTH     */
/*  LatitudeHemisphere = "1" = SOUTH     */
/*****************************************/
void Aircraft::SetLatitudeHemisphere(int h)
{   LatitudeHemisphere = h;  }

/*****************************************/
/*   SET LONGITUDE DEGREE                */
/*****************************************/
void Aircraft::SetLongitudeDegree(int ld)
{   LongitudeDegree = ld;}

/*****************************************/
/*   SET LONGITUDE MINUTE                */
/*****************************************/
void Aircraft::SetLongitudeMinute(int lm)
{   LongitudeMinute = lm;}

/*****************************************/
/*   SET LONGITUDE SECOND                */
/*****************************************/
void Aircraft::SetLongitudeSecond(double ls)
{   LongitudeSecond = ls;}

/*****************************************/
/*   SET VELOCITY X (ECEF FRAME)         */
/*****************************************/
void Aircraft::SetVelocityX(double vel)
{   VelocityX = vel;}

/*****************************************/
/*   SET VELOCITY Y (ECEF FRAME)         */
/*****************************************/
void Aircraft::SetVelocityY(double vel)
{   VelocityY = vel;}

/*****************************************/
/*   SET VELOCITY Z (ECEF FRAME)         */
/*****************************************/
void Aircraft::SetVelocityZ(double vel)
```

```
{   VelocityZ = vel;}

/***************************************/
/*   SET ALTITUDE                      */
/***************************************/
void Aircraft::SetAltitude(double alt)
{   Altitude = alt;  }




/***************************************/
/*   GET LATITUDE DEGREE               */
/***************************************/
int Aircraft::GetLatitudeDegree()
{  return LatitudeDegree;  }

/***************************************/
/*   GET LATITUDE MINUTE               */
/***************************************/
int Aircraft::GetLatitudeMinute()
{  return LatitudeMinute;  }

/***************************************/
/*   GET LATITUDE SECOND               */
/***************************************/
double Aircraft::GetLatitudeSecond()
{  return LatitudeSecond;  }

/***************************************/
/*   GET LATITUDE HEMISPHERE           */
/*  LatitudeHemisphere = "0" = NORTH   */
/*  LatitudeHemisphere = "1" = SOUTH   */
/***************************************/
int Aircraft::GetLatitudeHemisphere()
{  return LatitudeHemisphere;  }

/***************************************/
/*   GET LONGITUDE DEGREE              */
/***************************************/
int Aircraft::GetLongitudeDegree()
{  return LongitudeDegree;  }

/***************************************/
/*   GET LONGITUDE MINUTE              */
/***************************************/
int Aircraft::GetLongitudeMinute()
{  return LongitudeMinute;  }

/***************************************/
/*   GET LONGITUDE SECOND              */
/***************************************/
double Aircraft::GetLongitudeSecond()
{  return LongitudeSecond;  }

/***************************************/
/*   GET VELOCITY X                    */
/***************************************/
double Aircraft::GetVelocityX()
{  return VelocityX;  }

/***************************************/
```

```
/*    GET VELOCITY Y                        */
/****************************************/
double Aircraft::GetVelocityY()
{  return VelocityY;  }

/****************************************/
/*    GET VELOCITY Z                        */
/****************************************/
double Aircraft::GetVelocityZ()
{  return VelocityZ;  }

/****************************************/
/*    GET ALTITUDE                          */
/****************************************/
double Aircraft::GetAltitude()
{  return Altitude;  }
```

D.2 ErrorStructure.cpp

```
/****************************************************************************/
/*  MODULE NAME:      ErrorStructure.cpp                                    */
/*  AUTHOR:           Captain David Vloedman                                */
/*  DATE CREATED:     July 25, 1998                                         */
/*                                                                          */
/*  PURPOSE:          This module of code houses the error structure which  */
/*                    will be used to hold and trap any error conditions that */
/*                    arise during the operation of the program.            */
/*                                                                          */
/*  COMPILER:         Borland C++ Builder3 Standard version                 */
/*                    This compiler should be used to compile and link.     */
/*                                                                          */
/****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#pragma hdrstop
#pragma package(smart_init)
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "ErrorStructure.h"
#include "LaserConstants.h"

/********************************************/
/*   CREATE THE ErrorStructure CONSTRUCTOR  */
/********************************************/

ErrorStructure::ErrorStructure() :
    CriticalErrorFound(0),
    WarningFound(0),
    ErrorsFound(0)
    {
    for (int i = 0; i<MAXERRORS; i++)
      { strcpy(ModuleList[i]," ");
        strcpy(ErrorList[i]," ");
        Severity[i] = 0;
      }
    }

/********************************************/
/*   CREATE THE ErrorStructure DESTRUCTOR   */
/********************************************/

ErrorStructure::~ErrorStructure()
    {
    }

/****************************************************************************/
/*                   ErrorStructure MANIPULATION FUNCTIONS                  */
/****************************************************************************/
/****************************************************************************/
/*  FUNCTION NAME:  AddError                                                */
/*  AUTHOR:         Captain David Vloedman                                  */
```

```
/*  DATE CREATED:    July 25, 1998                                         */
/*                                                                         */
/*  PURPOSE:         This function is used to record an error into the error */
/*                   structure.                                            */
/***************************************************************************/
void ErrorStructure::AddError(char moduleName[MAXNAMELENGTH],
                              char description[MAXMESSAGELENGTH],
                              int   severity)


/**********************************************/
/*  MAKE CERTAIN THAT WE ARE NOT ADDING MORE  */
/*  ERRORS THAN THE MAX ALLOWED               */
/**********************************************/
{   if (ErrorsFound < (MAXERRORS - 1))
    {
        strcpy(ModuleList[ErrorsFound],moduleName);    /*******************/
        strcpy(ErrorList[ErrorsFound],description);    /* RECORD ERROR... */
        Severity[ErrorsFound] = severity;              /*******************/
        if (severity == 1)
            CriticalErrorFound = 1;
        else
            WarningFound = 1;
        ErrorsFound = ErrorsFound + 1;
    }

    /******************************************/
    /*  IF THERE HAVE ALREADY BEEN TOO MANY   */
    /*  ERRORS, SAY SO IN THE LAST ERROR IN   */
    /*  THE LIST                              */
    /******************************************/
    else
    {
        strcpy(ModuleList[MAXERRORS - 1],"Main Project");
        strcpy(ErrorList[MAXERRORS - 1],
               "Too Many Errors! Max number of errors Exceeded!");
        Severity[MAXERRORS - 1] = 1;
    }
}


/***************************************************************************/
/*  FUNCTION NAME:   GrabError                                             */
/*  AUTHOR:          Captain David Vloedman                                */
/*  DATE CREATED:    July 25, 1998                                         */
/*                                                                         */
/*  PURPOSE:         This function is used to retrieve an error that has been*/
/*                   previously added to the error structure. This routine  */
/*                   asks for the "number" of the error to grab (in order of */
/*                   when it was encountered) and grabs the information     */
/*                   associated with that error.                           */
/***************************************************************************/
void ErrorStructure::GrabError(int    number,
                               char   moduleName[MAXNAMELENGTH],
                               char   description[MAXMESSAGELENGTH],
                               int    &severity,
                               int    &found)
/*************************************/
/*  MAKE CERTAIN THAT THE ERROR THAT */
/*  IS CALLED FOR ACTUALLY EXISTS    */
/*************************************/
{   if (number <= ErrorsFound)
```

```
    {
        strcpy(moduleName, ModuleList[number-1]);
        strcpy(description, ErrorList[number-1]);
        severity = Severity[number-1];
        found = 1;
    }
    else
    /***************************************/
    /*  OTHERWISE TELL USER THAT ERROR     */
    /*  DOES NOT EXIST                     */
    /***************************************/
    {
        strcpy(moduleName,"Unknown");
        strcpy(description,"Unknown");
        severity = 0;
        found = 0;
    }
}


/****************************************************************************/
/*  FUNCTION NAME:  CriticalError                                           */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   July 25, 1998                                           */
/*                                                                          */
/*  PURPOSE:        This function is used to determine if a critical (fatal)*/
/*                  error has been detected and recorded yet.               */
/*                  CriticalErrorFound = 1 --> TRUE                         */
/*                  CriticalErrorFound = 0 --> FALSE                        */
/*                                                                          */
/****************************************************************************/
int ErrorStructure::CriticalError()
{   return CriticalErrorFound;  }


/****************************************************************************/
/*  FUNCTION NAME:  WarningError                                            */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   July 25, 1998                                           */
/*                                                                          */
/*  PURPOSE:        This function is used to determine if a warning (non-   */
/*                  fatal) error has been detected and recorded yet.        */
/*                  WarningFound = 1 --> TRUE                               */
/*                  WarningFound = 0 --> FALSE                              */
/*                                                                          */
/****************************************************************************/
int ErrorStructure::WarningError()
{   return WarningFound;  }


/****************************************************************************/
/*  FUNCTION NAME:  TotalErrors                                             */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   July 25, 1998                                           */
/*                                                                          */
/*  PURPOSE:        This function is used to determine how many errors total*/
/*                  have occurred and been recorded.                        */
/*                  ErrorsFound = Total number of errors.                   */
/*                                                                          */
/****************************************************************************/
int ErrorStructure::TotalErrors()
{   return ErrorsFound; }
```

```
/***************************************************************************/
/*  FUNCTION NAME:   CreateDisplayText                                     */
/*  AUTHOR:          Captain David Vloedman                                */
/*  DATE CREATED:    July 25, 1998                                         */
/*                                                                         */
/*  PURPOSE:         This function is used to create a simple array of     */
/*                   character arrays which hold all of the information    */
/*                   held in the error-structure.  This two-dimensional    */
/*                   text array may have messages as long as MAXMESSAGELENGTH*/
/*                   and can hold MAXERRORS messages.                      */
/*                                                                         */
/***************************************************************************/
void CreateDisplayText(ErrorStructure &errors,
                       char text[MAXERRORS][MAXMESSAGELENGTH])
{
    int i;
    int NumErrors = 0;
    int severe = 0;
    int found = 0;
    char module[MAXNAMELENGTH] = " ";
    char desc[MAXMESSAGELENGTH] = " ";
    char buff[MAXMESSAGELENGTH] = " ";


    NumErrors = errors.TotalErrors();
    /*****************************************/
    /*  GO THROUGH EACH ERROR                */
    /*****************************************/
    for (i = 1; i <= NumErrors; i++)
    {
    /*****************************************/
    /*  GRAB INFO FOR EACH ERROR             */
    /*****************************************/
        errors.GrabError(i, module, desc, severe, found);

        if (found)
        {

    /*****************************************/
    /*  IF THE ERROR IS A FATAL ERROR...     */
    /*****************************************/
            if (severe)
            {
                strcpy(buff,"Fatal Error: ");
                strcat(buff,module);
                strcat(buff,": ");
                strcat(buff,desc);
                strcpy(text[i-1],buff);
            }
    /*****************************************/
    /*  OTHERWISE IF THE ERROR IS A WARNING...*/
    /*****************************************/
            else
            {
                strcpy(buff,"Warning:      ");
                strcat(buff,module);
                strcat(buff,": ");
                strcat(buff,desc);
                strcpy(text[i-1],buff);
            }
        }
        else
```

```
            strcpy(text[i-1],"Warning:  Error list not found.");
    }
}
```

D.3 EvaluateEphemerisModules.cpp

```
/****************************************************************************/
/*  MODULE NAME:     EvaluateEphemerisModules.cpp
*/
/*  AUTHOR:          Captain David Vloedman                                  */
/*  DATE CREATED:    August 18, 1998                                         */
/*                                                                           */
/*  PURPOSE:         This set of modules supports the preprocessor and are   */
/*                   used to evaluate whether or not the satellite is ever   */
/*                   above the platform horizon.                             */
/*                                                                           */
/*  COMPILER:        Borland C++ Builder3 Standard version                   */
/*                   This compiler should be used to compile and link.       */
/*                                                                           */
/****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "TimeModules.h"
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisModules.h"
#include "SGP4SupportModules.h"
/**********************************/
/* C STANDARD LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/****************************************************************************/
/***********************        FUCTIONS       ******************************/
/****************************************************************************/

/****************************************************************************/
/*  FUNCTION NAME:  EvaluateEphemeris                                        */
/*  AUTHOR:         Captain David Vloedman                                   */
/*  DATE CREATED:   Sept 19, 1998                                            */
/*                                                                           */
/*  PURPOSE:        This function will take the position of the aircraft and*/
/*                  the orbital elements of the satellite and calculate      */
/*                  whether or not the satellite ever comes into view (or    */
/*                  above the horizontal horizon) of the the aircraft.       */
/*                                                                           */
/*  INPUTS:         NAME:                  DEFINITION:                       */
/*                  Sat                    Holds all ephemeris information */
/*                                         for the Satellite being studied */
/*                  ABLPlatform            Holds all information about ABL */
/*                                         Platform position/disposition   */
```

```
/*                  JulianDate           The time to which the position   */
/*                                       of sat should be propagated to   */
/*                  TimeToNextRun        The amount of time for which the*/
/*                                       current run must last.  This is */
/*                                       To determine how much time in    */
/*                                       seconds will transpire before    */
/*                                       next update is received.         */
/*                  ThetaGInRadians      The angle between the Greenwich */
/*                                       Meridian and the Vernal Equinox */
/*                                       at JulianDate.                   */
/*   OUTPUTS:       NAME:                DESCRIPTION:                     */
/*                  SatelliteInView      If the Satellite is visible to   */
/*                                       the ABLPlatform (over the        */
/*                                       artificial horizon of the        */
/*                                       aircraft. 1 = "yes", 0 = "no"    */
/*                  OrbitInView          Is the satellite ever above the */
/*                                       horizon plain of the platform?  */
/*                                       (IE, is the orbit itself, regard*/
/*                                       less of the satellite present    */
/*                                       position, it view? YES=1, NO=0. */
/*                  SatX                 X axis pos in ECI frame at Jul   */
/*                                       date                             */
/*                  SatY                 Y axis pos in ECI frame at Jul   */
/*                                       date                             */
/*                  SatZ                 Z axis pos in ECI frame at Jul   */
/*                                       date                             */
/*                  SatXdot              Velocity vector in X direction   */
/*                  SatYdot              Velocity vector in Y direction   */
/*                  SatZdot              Velocity vector in Z direction   */
/*                  Inclination          Inclination at Julian Date       */
/*                  RightAscension       Right Ascension at Julian Date   */
/*                  Eccentricity         Eccentricity at Julian Date      */
/*                  ArgumentOfPerigee    Arg of Perigee at Julian Date    */
/*                  Mean Anomaly         The Mean Anomanly at Julian Date*/
/*                  Delta                The amount of time in seconds    */
/*                                       that has transpired between the */
/*                                       actual ephemeris measurements    */
/*                                       and the Julian Date propagated   */
/*                  Dvector              This is the magnitude of the     */
/*                                       satellite radius vector (the     */
/*                                       vector from earth center to the */
/*                                       satellite) in the direction of   */
/*                                       the Platform radius vector.  IE */
/*                                       the component of the sat radius */
/*                                       vector in the Platform radius    */
/*                                       direction.  This is used to show*/
/*                                       how close the sat is to rising   */
/*                                       above the artificial horizon.    */
/*                  TimeToRise           Estimated time before the sat    */
/*                                       rises above the platform's       */
/*                                       artificial horizon.              */
/*                  CriticalRadius       The Radial component which tells*/
/*                                       the minimum distance an object   */
/*                                       must be before it lies above the*/
/*                                       artificial horizon of the        */
/*                                       platform.                        */
/*                  SatRadius            The Radial altitude of the sat   */
/*                                       wrt the platform altitude.  This*/
/*                                       is compared to the critical rad */
/*                                       to determine if the sat lies     */
/*                                       above or below the platform      */
/*                                       artificial horizon.              */
/*                  ErrorList            The Errors which have occurred   */
```

```
/*                                                                        */
/*  COMPILER:        Borland C++ Builder3 Standard version                */
/*                   This compiler should be used to compile and link.    */
/*                                                                        */
/**************************************************************************/
void EvaluateEphemeris( struct Satellite &Sat,
                        struct Aircraft &Platform,
                        double ThetaGInRad,
                        double JulianDate,
                        double TimeToNextRun,
                        int    &SatelliteInView,
                        int    &OrbitInView,
                        double &SatX,
                        double &SatY,
                        double &SatZ,
                        double &SatXdot,
                        double &SatYdot,
                        double &SatZdot,
                        double &Delta,
                        double &Inclination,
                        double &RightAscension,
                        double &Eccentricity,
                        double &MeanMotion,
                        double &ArgumentOfPerigee,
                        double &MeanAnomaly,
                        double &Dvector,
                        double &TimeToRise,
                        double &CriticalRadius,
                        double &SatRadius,
                        ErrorStructure   &ErrorList)
{
    double  Latitude;
    double  Longitude;
    double  LatInRadians;
    double  LonInRadians;

    double RaircraftECF[3];
    double RaircraftECI[3];
    double VaircraftECF[3];
    double VaircraftECI[3];
    double AircraftRadius;
    double MagnitudeRaircraftECI;
    double UnitRaircraftECI[3];
    double RsatECI[3];
    double VsatECI[3];
    double ChangeInD;
    char    buffer[MAXMESSAGELENGTH] = " ";
    Satellite *CurrentSat;
        CurrentSat = new Satellite;

/****************************************************/
/*   ERROR CHECK EACH INPUT PARAMETER FOR ERRORS    */
/****************************************************/
    if (Platform.GetAltitude() < 0)
    {   sprintf(buffer,"ABL Platform Altitude is very low -> %d",
                    Platform.GetAltitude());
        ErrorList.AddError("EvaluateEphemeris",
                            buffer,
                             0);
    }
    if ((Platform.GetLatitudeHemisphere() != 0) &&
        (Platform.GetLatitudeHemisphere() != 1))
    {    ErrorList.AddError("EvaluateEphemeris",
```

```
                         "Latitude Hemisphere must be north(N) or south(S)",
                          1);
}
if (Platform.GetLatitudeDegree() < 0)
{   sprintf(buffer,"Platform Latitude, %d, must be positive",
                 Platform.GetLatitudeDegree());
    ErrorList.AddError("EvaluateEphemeris",
                         buffer,
                          1);
}
if (Platform.GetLatitudeDegree() > 90)
{   sprintf(buffer,"Platform Latitude, %d, must be less than 90 degrees",
                 Platform.GetLatitudeDegree());
    ErrorList.AddError("EvaluateEphemeris",
                         buffer,
                          1);
}
if (Platform.GetLatitudeMinute() < 0)
{   sprintf(buffer,"Platform Latitude minutes, %d, must be positive",
                 Platform.GetLatitudeMinute());
    ErrorList.AddError("EvaluateEphemeris",
                         buffer,
                          1);
}
if (Platform.GetLatitudeMinute() > 60)
{   sprintf(buffer,"Platform Latitude minutes, %d, must be less than 60",
                 Platform.GetLatitudeMinute());
    ErrorList.AddError("EvaluateEphemeris",
                         buffer,
                          1);
}
if (Platform.GetLatitudeSecond() < 0)
{   sprintf(buffer,"Platform Latitude seconds, %d, must be positive",
                 Platform.GetLatitudeSecond());
    ErrorList.AddError("EvaluateEphemeris",
                         buffer,
                         1);
}
if (Platform.GetLatitudeSecond() > 60)
{   sprintf(buffer,"Platform Latitude seconds, %d, must be less than 60",
                 Platform.GetLatitudeSecond());
    ErrorList.AddError("EvaluateEphemeris",
                         buffer,
                         1);
}
if (Platform.GetLongitudeDegree() < 0)
{   sprintf(buffer,"Platform Longitude Deg, %d, must be positive deg East",
                 Platform.GetLongitudeDegree());
    ErrorList.AddError("EvaluateEphemeris",
                         buffer,
                         1);
}
if (Platform.GetLongitudeDegree() > 360)
{   sprintf(buffer,"Platform Longitude Deg, %d, must be < 360",
                 Platform.GetLongitudeDegree());
    ErrorList.AddError("EvaluateEphemeris",
                         buffer,
                         1);
}
if (Platform.GetLongitudeMinute() < 0)
{   sprintf(buffer,"Platform Longitude Min, %d, must be positive",
                 Platform.GetLongitudeMinute());
    ErrorList.AddError("EvaluateEphemeris",
```

```
                        buffer,
                        1);
}
if (Platform.GetLongitudeMinute() > 60)
{   sprintf(buffer,"Platform Longitude Min, %d, must be < 60",
                Platform.GetLongitudeMinute());
    ErrorList.AddError("EvaluateEphemeris",
                        buffer,
                        1);
}
if (Platform.GetLongitudeSecond() < 0)
{   sprintf(buffer,"Platform Longitude Sec, %d, must be positive",
                Platform.GetLongitudeSecond());
    ErrorList.AddError("EvaluateEphemeris",
                        buffer,
                        1);
}
if ((Platform.GetVelocityX() == 0.0) &&
    (Platform.GetVelocityY() == 0.0) &&
    (Platform.GetVelocityZ() == 0.0))
{   sprintf(buffer,"Platform is not moving, velocity is zero");
    ErrorList.AddError("EvaluateEphemeris",
                        buffer,
                        0);
}
if (Sat.GetRightAscension() < 0)
{   sprintf(buffer,"Satellite SSC: %d, has negative Right Ascension",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                        buffer,
                        1);
}
if (Sat.GetRightAscension() > 360)
{   sprintf(buffer,"Satellite SSC: %d, has Right Ascension > 360 deg",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                        buffer,
                        1);
}
if (Sat.GetEpochDay() < 0)
{   sprintf(buffer,"Satellite SSC: %d, has an Epoch Day < 0",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                        buffer,
                        1);
}
if (Sat.GetEpochDay() > 366)
{   sprintf(buffer,"Satellite SSC: %d, has an Epoch Day > 366",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                        buffer,
                        1);
}
if (Sat.GetEpochYear() < 1950)
{   sprintf(buffer,"Satellite SSC: %d, has an Epoch Year < 1950!",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                        buffer,
                        0);
}
if (Sat.GetMeanAnomaly() < 0)
{   sprintf(buffer,"Satellite SSC: %d, has a Mean Anomaly < 0",
                Sat.GetSSCNumber());
```

```
        ErrorList.AddError("EvaluateEphemeris",
                           buffer,
                           1);
}
if (Sat.GetMeanAnomaly() > 360)
{   sprintf(buffer,"Satellite SSC: %d, has a Mean Anomaly > 360 deg",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                       buffer,
                       1);
}
if (Sat.GetInclination() < 0)
{   sprintf(buffer,"Satellite SSC: %d, has an Inclination < 0",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                       buffer,
                       1);
}
if (Sat.GetInclination() > 180)
{   sprintf(buffer,"Satellite SSC: %d, has an Inclination > 180 deg",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                       buffer,
                       1);
}
if (Sat.GetEccentricity() < 0)
{   sprintf(buffer,"Satellite SSC: %d, has an Eccentricity < 0",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                       buffer,
                       1);
}
if (Sat.GetEccentricity() >= 1)
{   sprintf(buffer,"Satellite SSC: %d, has an Eccentricity > 1.0",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                       buffer,
                       1);
}
if (Sat.GetArgumentOfPerigee() < 0)
{   sprintf(buffer,"Satellite SSC: %d, has an Argument of Perigee < 0",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                       buffer,
                       1);
}
if (Sat.GetArgumentOfPerigee() > 360)
{   sprintf(buffer,"Satellite SSC: %d, has an Argument of Per > 360 deg",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                       buffer,
                       1);
}
if (Sat.GetMeanMotion() <= 0)
{   sprintf(buffer,"Mean Motion <= 0.0 for Satellite SSC: %d",
                Sat.GetSSCNumber());
    ErrorList.AddError("EvaluateEphemeris",
                       buffer,
                       1);
}
if (TimeToNextRun <= 0.0)
{    ErrorList.AddError("EvaluateEphemeris",
                       "The time until the next run cannot be <= 0.0 sec",
```

```
                              1);
    }
    if (TimeToNextRun > 300.0)
    {    ErrorList.AddError("EvaluateEphemeris",
    "HIGHLY RECOMMEND that Preprocessor be run at LEAST every 300 seconds.",
                              0);
    }

/************************************************/
/*   INITIALIZE OUTPUT VARIABLES                */
/************************************************/
    SatelliteInView = 0;
    OrbitInView = 0;
    SatX = 0.0;
    SatY = 0.0;
    SatZ = 0.0;
    SatXdot = 0.0;
    SatYdot = 0.0;
    SatZdot = 0.0;
    Delta = 0.0;
    Inclination = 0.0;
    RightAscension = 0.0;
    Eccentricity = 0.0;
    MeanMotion = 0.0;
    ArgumentOfPerigee = 0.0;
    MeanAnomaly = 0.0;
    Dvector = 0.0;
    TimeToRise = 0.0;
    CriticalRadius = 0.0;
    SatRadius = 0.0;

/************************************************/
/*   BEGIN CALCULATIONS UNLESS CRITICAL ERROR   */
/************************************************/
    if (ErrorList.CriticalError())
        return;

/************************************************/
/*   FIND LAT AND LON IN RADIANS                */
/*   NOTE THAT -LAT = SOUTHERN LATITUDE         */
/*   LatitudeHemisphere = "0" = NORTH LAT       */
/*   LatitudeHemisphere = "1" = SOUTH LAT       */
/************************************************/
    Latitude =  (Platform.GetLatitudeDegree()) +
                (Platform.GetLatitudeMinute()/60.0) +
                (Platform.GetLatitudeSecond()/3600.0);
    LatInRadians = Latitude * DEGTORADIANS;
    if (Platform.GetLatitudeHemisphere() == 1)
          LatInRadians = -LatInRadians;

    if (Latitude < -90.0)
    {    ErrorList.AddError("EvaluateEphemeris",
                            "Latitude of platform is more than 90 deg south",
                             1);
    }
    if (Latitude > 90.0)
    {    ErrorList.AddError("EvaluateEphemeris",
                            "Latitude of platform is more than 90 deg north",
                             1);
    }


    Longitude = (Platform.GetLongitudeDegree()) +
```

```
                (Platform.GetLongitudeMinute()/60.0) +
                (Platform.GetLongitudeSecond()/3600.0);
    LonInRadians = Longitude * DEGTORADIANS;
    if (Longitude > 360.0)
    {    ErrorList.AddError("EvaluateEphemeris",
                            "Longitude of platform is > 360 deg",
                             1);
    }


/************************************************/
/*   CONVERT LATITUDE, LONGITUDE AND ALTITUDE    */
/*   POSITION OF THE AIRCRAFT TO A RADIAL VECTOR*/
/*   IN THE EARTH-CENTERED EARTH-FIXED COORD.    */
/*   FRAME                                       */
/*     RaircraftECF[0] = X                       */
/*     RaircraftECF[1] = Y                       */
/*     RaircraftECF[2] = Z                       */
/************************************************/
    AircraftRadius = EARTHRADIUS + Platform.GetAltitude();

    RaircraftECF[0] = AircraftRadius *
                      cos(LatInRadians) *
                      cos(LonInRadians);
    RaircraftECF[1] = AircraftRadius *
                      cos(LatInRadians) *
                      sin(LonInRadians);
    RaircraftECF[2] = AircraftRadius *
                      sin(LatInRadians);

/************************************************/
/*   CONVERT EARTH-CENTERED EARTH-FIXED COORD.   */
/*  FRAME TO EARTH-CENTERED-INERTIAL BY USING    */
/*  THETA-G AS THE ROTATION ANGLE.               */
/*     RaircraftECI[0] = X                       */
/*     RaircraftECI[1] = Y                       */
/*     RaircraftECI[2] = Z                       */
/************************************************/
    RaircraftECI[0] = RaircraftECF[0] * cos(ThetaGInRad) -
                      RaircraftECF[1] * sin(ThetaGInRad);
    RaircraftECI[1] = RaircraftECF[0] * sin(ThetaGInRad) +
                      RaircraftECF[1] * cos(ThetaGInRad);
    RaircraftECI[2] = RaircraftECF[2];


/************************************************/
/*   FIND VELOCITY OF THE AIRCRAFT VECTOR        */
/*   IN THE EARTH-CENTERED EARTH-FIXED COORD.    */
/*   FRAME                                       */
/*     VaircraftECF[0] = Xdot                    */
/*     VaircraftECF[1] = Ydot                    */
/*     VaircraftECF[2] = Zdot                    */
/************************************************/
    VaircraftECF[0] = Platform.GetVelocityX();
    VaircraftECF[1] = Platform.GetVelocityY();
    VaircraftECF[2] = Platform.GetVelocityZ();

/************************************************/
/*   CONVERT EARTH-CENTERED EARTH-FIXED COORD.   */
/*  FRAME TO EARTH-CENTERED-INERTIAL BY USING    */
/*  THETA-G AS THE ROTATION ANGLE.  NOTE THAT    */
/*  THIS CAPTURES THE ROTATION OF THE EARTH      */
/*  UNDERNEATH THE PLANE.                        */
```

```
/*      VaircraftECI[0] = Xdot                         */
/*      VaircraftECI[1] = Ydot                         */
/*      VaircraftECI[2] = Zdot                         */
/*   THE UNITS HERE IN THE ECI FRAME ARE:              */
/*       KILOMETERS * RADIANS / HOURS                  */
/*************************************************/
    VaircraftECI[0] = VaircraftECF[0] * cos(ThetaGInRad) -
                      VaircraftECF[1] * sin(ThetaGInRad) -
                      RaircraftECI[1] * TWOPI/(SECSSIDEREALDAY/3600);
    VaircraftECI[1] = VaircraftECF[0] * sin(ThetaGInRad) +
                      VaircraftECF[1] * cos(ThetaGInRad) +
                      RaircraftECI[0] * TWOPI/(SECSSIDEREALDAY/3600);
    VaircraftECI[2] = VaircraftECF[2];

/*************************************************/
/*   FIND THE UNIT VECTOR IN THE DIRECTION OF THE */
/*   PLATFORM POSITION VECTOR.  THIS IS USED TO   */
/*   THE MAGNITUDE OF COMPONENTS OF OTHER VECTORS */
/*   IN THE DIRECTION OF THE PLATFORM POSITION    */
/*   VECTOR.                                      */
/*************************************************/
    MagnitudeRaircraftECI = sqrt(pow(RaircraftECI[0],2) +
                                 pow(RaircraftECI[1],2) +
                                 pow(RaircraftECI[2],2));

    if (MagnitudeRaircraftECI != 0.0)
    {
        UnitRaircraftECI[0] = RaircraftECI[0] / MagnitudeRaircraftECI;
        UnitRaircraftECI[1] = RaircraftECI[1] / MagnitudeRaircraftECI;
        UnitRaircraftECI[2] = RaircraftECI[2] / MagnitudeRaircraftECI;
    }
    else
    {    ErrorList.AddError("EvaluateEphemeris",
                            "Magnitude of aircraft position vector is 0.0",
                             1);
    }

/*************************************************/
/*   FIND THE POSITION AND VELOCITY VECTORS OF THE*/
/*   SATELLITE FOR THE GIVEN PROPAGATION TIME     */
/*   (WHICH IS STORED IN "JulianDate").           */
/*   NOTE: SGP4 CANNOT HANDLE A PERFECTLY ROUND   */
/*   EPHEMERIS (IE Eccentricity CANNOT EQUAL 0.0  */
/*************************************************/
    if (Sat.GetEccentricity() == 0)
    {   sprintf(buffer,"Satellite SSC: %d, has an Eccent = 0.0, SGP4 Error",
                    Sat.GetSSCNumber());
        ErrorList.AddError("EvaluateEphemeris",
                            buffer,
                            1);
        return;
    }
    CallSGP4(Sat,
             JulianDate,
             SatX,
             SatY,
             SatZ,
             SatXdot,
             SatYdot,
             SatZdot,
             Inclination,
             RightAscension,
             Eccentricity,
```

```
            MeanMotion,
            ArgumentOfPerigee,
            MeanAnomaly,
            Delta,
            ErrorList);

/************************************************/
/*   CONTINUE UNLESS CRITICAL ERROR            */
/************************************************/
    if (ErrorList.CriticalError())
        return;

/************************************************/
/*  HERE, I AM SIMPLY MOVING THE PARAMETERS TO   */
/*  A MATRIX.  THIS COULD HAVE BEEN DONE WITH A  */
/*  LOT OF SHORTCUTS, BUT I DO IT THIS LONG WAY  */
/*  TO ENHANCE READABILITY OF THE PROGRAM AS MUCH*/
/*  AS POSSIBLE.                                 */
/************************************************/
    RsatECI[0] = SatX;
    RsatECI[1] = SatY;
    RsatECI[2] = SatZ;

    VsatECI[0] = SatXdot;
    VsatECI[1] = SatYdot;
    VsatECI[2] = SatZdot;

/************************************************/
/*  THE Dvector IS THE COMPONENT OF THE SAT     */
/*  POSITION VECTOR IN THE PLATFORM POSITION    */
/*  VECTOR DIRECTION.  THIS IS USED TO SEE HOW  */
/*  CLOSE THE SATELLITES POSITION COMPARES TO   */
/*  THE PLATFORM'S ARTIFICIAL HORIZON, WHICH IS */
/*  SIMPLY THE PLANE PERPENDICULAR TO THE       */
/*  PLATFORM POSITION VECTOR. (ASSUME STRAIGHT  */
/*  AND LEVEL FLIGHT).                          */
/************************************************/
    Dvector = RsatECI[0] * UnitRaircraftECI[0] +
              RsatECI[1] * UnitRaircraftECI[1] +
              RsatECI[2] * UnitRaircraftECI[2];

/************************************************/
/*  IF THE Dvector IS HAS GREATER LENGTH THAN THE*/
/*  PLATFORM POSITION VECTOR, THEN WE KNOW THAT */
/*  THE SATELLITE LIES ABOVE THE PLATFORM'S     */
/*  ARTIFICIAL HORIZON, AND IS THEREFORE IN VIEW */
/*  (LINE-OF-SIGHT) OF THE PLATFORM.            */
/************************************************/

    if (Dvector >= AircraftRadius)
    {   SatelliteInView = 1;
        OrbitInView = 1;
        TimeToRise = 0.0;
    }
    else
/************************************************/
/*  IF THE PLATFORM IS NOT YET IN VIEW, THEN    */
/*  DETERMINE WHEN, IF EVER, THE PLATFORM DOES  */
/*  COME INTO VIEW.  IF THE SATELLITE IS ABOUT  */
/*  TO COME INTO VIEW BEFORE THE "TimeToNextRun" */
/*  OF THE Preprocessor, THEN INCLUDE THIS SAT  */
/*  AS BEING IN VIEW.                           */
/************************************************/
```

```
    {   SatelliteInView = 0;
        OrbitInView = 0;

/**************************************************/
/*  LOAD ANOTHER Satellite DATA STRUCTURE WITH    */
/*  THE CURRENT EPHEMERIS INFORMATION GLEANED     */
/*  THE TIME PROPAGATOR (Sgp4) AND FEED IT TO     */
/*  CompareOrbit TO SEE IF THE CURRENT ORBIT      */
/*  WILL CROSS THE HORIZON OF THE PLATFORM        */
/**************************************************/
        CurrentSat->SetInclination(Inclination);
        CurrentSat->SetRightAscension(RightAscension);
        CurrentSat->SetEccentricity(Eccentricity);
        CurrentSat->SetArgumentOfPerigee(ArgumentOfPerigee);
        CurrentSat->SetMeanAnomaly(MeanAnomaly);

        MeanMotion = MeanMotion * MINUTESPERDAY / TWOPI;
        CurrentSat->SetMeanMotion(MeanMotion);

        CompareOrbit(*CurrentSat,
                     Platform,
                     ThetaGInRad,
                     OrbitInView,
                     CriticalRadius,
                     SatRadius,
                     ErrorList);

/**************************************************/
/*  IF THE ORBIT IS IN VIEW, THEN FIND THE        */
/*  APPROXIMATE TIME UNTIL THE SATELLITE BREAKS   */
/*  THROUGH THE HORIZON OF THE PLATFORM.  THIS    */
/*  IS DONE ONLY AS AN APPROXIMATION.             */
/**************************************************/
        if (OrbitInView)
        {
            ChangeInD = VsatECI[0] * SECSPERHOUR * UnitRaircraftECI[0] +
                        VsatECI[1] * SECSPERHOUR * UnitRaircraftECI[1] +
                        VsatECI[2] * SECSPERHOUR * UnitRaircraftECI[2] +
                        RsatECI[0] * VaircraftECI[0] / AircraftRadius +
                        RsatECI[1] * VaircraftECI[1] / AircraftRadius +
                        RsatECI[2] * VaircraftECI[2] / AircraftRadius;

            if (ChangeInD != 0.0)
            {
                TimeToRise = (Dvector - AircraftRadius) / ChangeInD;
                TimeToRise = TimeToRise * SECSPERHOUR;

/**************************************************/
/*  IF THE TimeToRise IS POSITIVE (THAT IS, IT IS*/
/*  MOVING "TOWARDS" THE PLATFORM, NOT AWAY) AND */
/*  THE TIME BEFORE THE NEXT RUN OF THE          */
/*  Preprocessor IS MORE THAN THE TimeToRise,    */
/*  THEN INCLUDE THIS SATELLITE AS ONE WHICH     */
/*  COULD BE IN VIEW BEFORE THE NEXT RUN.        */
/**************************************************/
                if ((TimeToNextRun > TimeToRise) && (TimeToRise > 0.0))
                    SatelliteInView = 1;
            }
            else
            {   ErrorList.AddError("EvaluateEphemeris",
                               "ChangeInD is zero",
                                1);
            }
```

```
        }
    }

    return;

}


/****************************************************************************/
/*  FUNCTION NAME:  FindThetaG                                              */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   October 6, 1998                                         */
/*                                                                          */
/*  PURPOSE:        This function will take a reference position and time   */
/*                  for a known angle between the Greenwich Meridian and    */
/*                  the Vernal Equinox, and propagate the angle through     */
/*                  natural orbit precession at the given calculation time. */
/*                  Note that the reference time must always be BEFORE the  */
/*                  calulation time.                                        */
/*                                                                          */
/*  INPUTS:         NAME:                   DEFINITION:                     */
/*                  ReferenceHour           This holds the value of Theta G */
/*                                          at RefModJulianDate.  The angle */
/*                                          of Theta G is given in hours,   */
/*                                          minutes, and seconds instead of */
/*                                          degrees, where 24 hrs = 360 deg */
/*                  ReferenceMinute         Holds the minutes of Theta G at */
/*                                          RefModJulianDate.               */
/*                  ReferenceSecond         Holds the seconds of Theta G at */
/*                                          RefModJulianDate.               */
/*                  RefModJulianDate        This is the reference date when */
/*                                          an actual observation of the    */
/*                                          true value of theta G was made. */
/*                  CalcYear                Holds the current calender year */
/*                  Calcmonth               Holds the Calender month(1 - 12)*/
/*                  CalcDay                 Holds calender day              */
/*                  CalcHour                Holds the calender hour         */
/*                  CalcMinute              Holds the calender minute       */
/*                  CalcSecond              Holds the calender second       */
/*                                                                          */
/*  OUTPUTS:        NAME:                   DESCRIPTION:                    */
/*                  ThetaGInRadians         The angle between the Greenwich */
/*                                          Meridian and the Vernal Equinox */
/*                                          at Calc Date.                   */
/*                  ErrorList               The Errors which have occurred  */
/*                                                                          */
/*  COMPILER:       Borland C++ Builder3 Standard version                   */
/*                  This compiler should be used to compile and link.       */
/*                                                                          */
/****************************************************************************/
void FindThetaG(int     ReferenceHour,
                int     ReferenceMinute,
                double  ReferenceSecond,
                double  RefModJulianDate,
                int     CalcYear,
                int     CalcMonth,
                int     CalcDay,
                int     CalcHour,
                int     CalcMinute,
                double  CalcSecond,
                double  &ThetaGInRadians,
                ErrorStructure   &ErrorList)
```

```
{
    double   RefThetaGInDeg;
    double   CurrentModJulianDate;
    double   PropTime;
    double   PropRate;
    double   ThetaGInDeg;
    double   RefThetaGInSec;


/**********************************************/
/*    ERROR CHECK INCOMING PARAMETERS         */
/**********************************************/
    if (ReferenceHour < 0)
    {    ErrorList.AddError("FindThetaG",
                           "Theta G Reference Hour < 0",
                            1);
    }
    if (ReferenceHour > 24)
    {    ErrorList.AddError("FindThetaG",
                           "Theta G Reference Hour > 24",
                            1);
    }
    if (ReferenceMinute < 0)
    {    ErrorList.AddError("FindThetaG",
                           "Theta G Reference Minute < 0",
                            1);
    }
    if (ReferenceMinute > 60)
    {    ErrorList.AddError("FindThetaG",
                           "Theta G Reference Minute > 60",
                            1);
    }
    if (ReferenceSecond < 0)
    {    ErrorList.AddError("FindThetaG",
                           "Theta G Reference Second < 0",
                            1);
    }
    if (ReferenceSecond > 60)
    {    ErrorList.AddError("FindThetaG",
                           "Theta G Reference Second > 60",
                            1);
    }
    if (RefModJulianDate < 0)
    {    ErrorList.AddError("FindThetaG",
                           "Reference Julian Date < 0",
                            1);
    }
    if (CalcYear < 0)
    {    ErrorList.AddError("FindThetaG",
                           "Calculation Year < 0",
                            1);
    }
    if (CalcYear > 3000)
    {    ErrorList.AddError("FindThetaG",
                           "Calculation Year > 3000",
                            1);
    }
    if (CalcMonth < 0)
    {    ErrorList.AddError("FindThetaG",
                           "Calculation Month < 0",
                            1);
    }
    if (CalcMonth > 12)
```

```
    {    ErrorList.AddError("FindThetaG",
                             "Calculation Month > 12",
                              1);
    }
    if (CalcDay < 0)
    {    ErrorList.AddError("FindThetaG",
                             "Calculation Day < 0",
                              1);
    }
    if (CalcDay > 31)
    {    ErrorList.AddError("FindThetaG",
                             "Calculation Day > 31",
                              1);
    }
    if (CalcHour < 0)
    {    ErrorList.AddError("FindThetaG",
                             "Calculation Hour < 0",
                              1);
    }
    if (CalcHour > 24)
    {    ErrorList.AddError("FindThetaG",
                             "Calculation Hour > 24",
                              1);
    }
    if (CalcMinute < 0)
    {    ErrorList.AddError("FindThetaG",
                             "Calculation Minute < 0",
                              1);
    }
    if (CalcMinute > 60)
    {    ErrorList.AddError("FindThetaG",
                             "Calculation Minute > 60",
                              1);
    }
    if (CalcSecond < 0)
    {    ErrorList.AddError("FindThetaG",
                             "Calculation Second < 0",
                              1);
    }
    if (CalcSecond > 60)
    {    ErrorList.AddError("FindThetaG",
                             "Calculation Second > 60",
                              1);
    }

/***********************************************/
/*   BEGIN CALCULATIONS UNLESS CRITICAL ERROR    */
/***********************************************/
    if (ErrorList.CriticalError())
        return;


/**********************************************/
/*  FIND REFERENCE THETA G IN DEGREES          */
/**********************************************/
    RefThetaGInSec = ((ReferenceHour * 3600) +
                      (ReferenceMinute * 60) +
                       ReferenceSecond);

    if (RefThetaGInSec > SECSPER24HOURS)
    {    ErrorList.AddError("FindThetaG",
                             "Reference Angle Exceeds 24 hours (360 degrees)",
                             1);
```

```
        return;
    }

    RefThetaGInDeg =  RefThetaGInSec * (360.0 / SECSPER24HOURS);

/***********************************************/
/*  GET CURRENT JULIAN DATE                    */
/***********************************************/
    ConvertCalenderToJulian(CalcYear,
                            CalcMonth,
                            CalcDay,
                            CalcHour,
                            CalcMinute,
                            CalcSecond,
                            CurrentModJulianDate,
                            ErrorList);
    if (ErrorList.CriticalError())
        return;

/***********************************************/
/*  DETERMINE THE PROPAGATION TIME             */
/***********************************************/
    PropTime = (CurrentModJulianDate - RefModJulianDate) * 24 *3600;;
    if (PropTime < 0.0)
    {    ErrorList.AddError("FindThetaG",
                        "Reference Time should occur before Calculation Time",
                            1);
        return;
    }
    if (PropTime > LATEREFERENCE)
    {    ErrorList.AddError("FindThetaG",
        "Ref Time and Calc Time are too far apart to safely predict ThetaG",
                            0);
        return;
    }

/***********************************************/
/*  DETERMINE THE PROPAGATION RATE             */
/*  NOTE THAT THE NUMBER OF SECONDS IN A       */
/*  SIDEREAL DAY = 23 hrs 56 mins 4.09054 secs*/
/*               = 86164.09054 secs            */
/*               = SECSSIDEREALDAY             */
/***********************************************/
    PropRate = 360 / SECSSIDEREALDAY;

/***********************************************/
/*  PROPAGATE THETA G THROUGH TIME             */
/***********************************************/
    ThetaGInDeg = RefThetaGInDeg + PropRate * PropTime;

/***********************************************/
/*  DIVIDE MOD 360 DEGREES TO GET CURRENT POS */
/***********************************************/
    ThetaGInDeg = fmod(ThetaGInDeg, 360.0);

/***********************************************/
/*  CONVERT TO RADIANS                         */
/***********************************************/
    ThetaGInRadians = ThetaGInDeg * DEGTORADIANS;

    return;
}
```

```
/****************************************************************************/
/*  FUNCTION NAME:  CompareOrbit                                            */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   October 6, 1998                                         */
/*                                                                          */
/*  PURPOSE:        This function will take the position of the aircraft and*/
/*                  the orbital elements of the satellite and calculate     */
/*                  whether or not the satellite ever comes into view (or   */
/*                  above the horizontal horizon) of the the aircraft.  Note*/
/*                  that this is at an instantaneous time.  It does not     */
/*                  account for the precession of the orbit, and so must    */
/*                  be run at regular close (30 minute) intervals to be     */
/*                  reliable and accurate.                                  */
/*                                                                          */
/*  INPUTS:         NAME:                   DEFINITION:                     */
/*          Platform.LatitudeDegree         Degree of Latitude (0-90 int)   */
/*          Platform.LatitudeMinute         Minute of Latitude (0-60 int)   */
/*          Platform.LatitudeSecond         Second of Latitude (0-60 float) */
/*          Platform.LongitudeDegree        Degree of Longitude (0-360 int) */
/*          Platform.LongitudeMinute        Minute of Longitude (0-60 int)  */
/*          Platform.LongitudeSecond        Second of Longitude (0-60 float)*/
/*          Sat.RightAscension              Right Ascension (degrees)       */
/*          Sat.Eccentricity                Eccentricity    (float)         */
/*          Sat.Inclination                 Inclination (degrees)           */
/*          Sat.MeanMotion                  Mean Motion (float)             */
/*          Sat.ArgumentOfPerigee           Degrees (0-360)                 */
/*          Sat.MeanAnomaly                 Degrees (0-360)                 */
/*          Sat.EpochDay                    Day of year msrmts taken (float)*/
/*          Sat.EpochYear                   Calender Year (int)             */
/*          ThetaGInRad                     Angle between Greenwich and     */
/*                                             Vernal Equinox               */
/*          ErrorList                       Errors that have occured        */
/*                                                                          */
/*  OUTPUTS:    NAME:                       DESCRIPTION:                    */
/*          CriticalRadius                  The Radial component which tells*/
/*                                          the minimum distance an object  */
/*                                          must be before it lies above the*/
/*                                          artificial horizon of the       */
/*                                          platform.                       */
/*          SatRadius                       The Radial altitude of the sat  */
/*                                          wrt the platform altitude.  This*/
/*                                          is compared to the critical rad */
/*                                          to determine if the sat lies    */
/*                                          above or below the platform     */
/*                                          artificial horizon.             */
/*          OrbitInView                     Is the satellite ever above the */
/*                                          horizon plain of the platform?  */
/*                                          (IE, is the orbit itself, regard*/
/*                                          less of the satellite present   */
/*                                          position, it view? YES=1, NO=0. */
/*                                                                          */
/*                                                                          */
/*  COMPILER:       Borland C++ Builder3 Standard version                   */
/*                  This compiler should be used to compile and link.       */
/*                                                                          */
/****************************************************************************/
void CompareOrbit( struct Satellite &Sat,
                   struct Aircraft &Platform,
                   double ThetaGInRad,
                   int    &OrbitInView,
                   double &CriticalRadius,
                   double &SatRadius,
```

```
                   ErrorStructure    &ErrorList)
{
    double  Latitude;
    double  Longitude;
    double  LatInRadians;
    double  LonInRadians;
    double  RAInRad;
    double  CosinePhi;
    double  InclinInRad;
    double  Phi;
    double  CosineAlpha;
    double  Alpha;
    double  Beta;
    double  D;
    double  SineOfVandW1;
    double  VandW1;
    double  CosineOfVandW2;
    double  VandW2;
    double  VandW;
    double  TrueAnomalyInRad;
    double  Numerator;
    double  Denominator;
    double  SemiMajorAxis;
    double  Eccentricity;
    double  Altitude;
    double  WInRad;
    double  SineD;
    double  X;
    double  Y;
    char    buffer[MAXMESSAGELENGTH] = " ";

/*************************************************/
/*   ERROR CHECK EACH PARAMETER                  */
/*************************************************/
    if (Platform.GetAltitude() < 0)
    {    ErrorList.AddError("CompareOrbit",
                           "Platform Altitude is below sealevel",
                            0);
    }
    if ((Platform.GetLatitudeHemisphere() != 0) &&
        (Platform.GetLatitudeHemisphere() != 1))
    {    ErrorList.AddError("CompareOrbit",
                           "Latitude Hemisphere must be north(N) or south(S)",
                            1);
    }
    if (Platform.GetLatitudeDegree() < 0)
    {    ErrorList.AddError("CompareOrbit",
                           "Latitude degree of platform must be positive",
                            1);
    }
    if (Platform.GetLatitudeDegree() > 90)
    {    ErrorList.AddError("CompareOrbit",
                           "Latitude degree of platform is greater than 90",
                            1);
    }
    if (Platform.GetLatitudeMinute() < 0)
    {    ErrorList.AddError("CompareOrbit",
                           "Latitude minute of platform is less than 0",
                            1);
    }
    if (Platform.GetLatitudeMinute() > 60)
    {    ErrorList.AddError("CompareOrbit",
                           "Latitude minute of platform is greater than 60",
```

```
                        1);
}
if (Platform.GetLatitudeSecond() < 0)
{    ErrorList.AddError("CompareOrbit",
                       "Latitude second of platform is less than 0",
                        1);
}
if (Platform.GetLatitudeSecond() > 60)
{    ErrorList.AddError("CompareOrbit",
                       "Latitude second of platform is greater than 60",
                        1);
}
if (Platform.GetLongitudeDegree() < 0)
{    ErrorList.AddError("CompareOrbit",
                       "Longitude degree of platform is less than 0",
                        1);
}
if (Platform.GetLongitudeDegree() > 360)
{    ErrorList.AddError("CompareOrbit",
                       "Longitude degree of platform is greater than 360",
                        1);
}
if (Platform.GetLongitudeMinute() < 0)
{    ErrorList.AddError("CompareOrbit",
                       "Longitude minute of platform is less than 0",
                        1);
}
if (Platform.GetLongitudeMinute() > 60)
{    ErrorList.AddError("CompareOrbit",
                       "Longitude minute of platform is greater than 60",
                        1);
}
if (Platform.GetLongitudeSecond() < 0)
{    ErrorList.AddError("CompareOrbit",
                       "Longitude second of platform is less than 0",
                        1);
}
if (Platform.GetLongitudeSecond() > 60)
{    ErrorList.AddError("CompareOrbit",
                       "Longitude second of platform is greater than 60",
                        1);
}
if (Sat.GetRightAscension() < 0)
{    ErrorList.AddError("CompareOrbit",
                       "Right ascension of satellite is less than 0",
                        1);
}
if (Sat.GetRightAscension() > 360)
{    ErrorList.AddError("CompareOrbit",
                       "Right ascension of satellite is > 360 degrees",
                        1);
}
if (Sat.GetMeanAnomaly() < 0)
{    ErrorList.AddError("CompareOrbit",
                       "Mean anomaly of satellite is less than 0",
                        1);
}
if (Sat.GetMeanAnomaly() > 360)
{    ErrorList.AddError("CompareOrbit",
                       "Mean anomaly of satellite is > 360 degrees",
                        1);
}
if (Sat.GetInclination() < 0)
```

```
    {   ErrorList.AddError("CompareOrbit",
                           "Inclination of satellite is less than 0",
                            1);
    }
    if (Sat.GetInclination() > 180)
    {   ErrorList.AddError("CompareOrbit",
                           "Inclination of satellite is > 180 degrees",
                            1);
    }
    if (Sat.GetEccentricity() < 0)
    {   ErrorList.AddError("CompareOrbit",
                           "Eccentricity of satellite is less than 0",
                            1);
    }
    if (Sat.GetEccentricity() >= 1)
    {   ErrorList.AddError("CompareOrbit",
                           "Eccentricity of satellite is >= 1",
                            1);
    }
    if (Sat.GetArgumentOfPerigee() < 0)
    {   ErrorList.AddError("CompareOrbit",
                           "Arg of Perigee of satellite is less than 0 deg",
                            1);
    }
    if (Sat.GetArgumentOfPerigee() > 360)
    {   ErrorList.AddError("CompareOrbit",
                           "Arg of perigee of satellite is > 360 degrees",
                            1);
    }
    if (Sat.GetMeanMotion() <= 0.0)
    {   sprintf(buffer,"Mean Motion <= 0.0 for Satellite SSC: %d",
                    Sat.GetSSCNumber());
        ErrorList.AddError("CompareOrbit",
                           buffer,
                           1);
    }

/*************************************************/
/*   BEGIN CALCULATIONS UNLESS CRITICAL ERROR    */
/*************************************************/
    if (ErrorList.CriticalError())
        return;

/*************************************************/
/*   FIND LAT AND LON IN RADIANS                 */
/*   NOTE THAT -LAT = SOUTHERN LATITUDE          */
/*   LatitudeHemisphere = "0" = NORTH LAT        */
/*   LatitudeHemisphere = "1" = SOUTH LAT        */
/*************************************************/
    Latitude =   (Platform.GetLatitudeDegree()) +
                 (Platform.GetLatitudeMinute()/60.0) +
                 (Platform.GetLatitudeSecond()/3600.0);
    LatInRadians = Latitude * DEGTORADIANS;
    if (Platform.GetLatitudeHemisphere() == 1)
          LatInRadians = -LatInRadians;

    if (Latitude < -90.0)
    {   ErrorList.AddError("CompareOrbit",
                           "Latitude of platform is more than 90 deg south",
                            1);
    }
    if (Latitude > 90.0)
    {   ErrorList.AddError("CompareOrbit",
```

```
                              "Latitude of platform is > 90 deg north",
                               1);
    }


    Longitude = (Platform.GetLongitudeDegree()) +
                (Platform.GetLongitudeMinute()/60.0) +
                (Platform.GetLongitudeSecond()/3600.0);
    LonInRadians = Longitude * DEGTORADIANS;
    if (Longitude > 360.0)
    {    ErrorList.AddError("CompareOrbit",
                              "Longitude of platform is > 360 deg",
                               1);
    }


/************************************************/
/*   CONTINUE UNLESS CRITICAL ERROR             */
/************************************************/
    if (ErrorList.CriticalError())
        return;


/************************************************/
/*   CONVERT RIGHT ASCENSION TO RADIANS         */
/************************************************/
    RAInRad = Sat.GetRightAscension() * DEGTORADIANS;

/************************************************/
/*   CONVERT INCLINATION TO RADIANS             */
/************************************************/
    InclinInRad = Sat.GetInclination() * DEGTORADIANS;

/************************************************/
/*   FIND PHI - THE SPHERICAL ANGLE BETWEEN THE */
/*   PLATFORM AND THE ASCENDING NODE OF THE     */
/*   SATELLITE EPHEMERIS                        */
/************************************************/
    CosinePhi = cos(LatInRadians) *
                cos(RAInRad - (ThetaGInRad + LonInRadians));

    Phi = acos(CosinePhi);

/************************************************/
/*   FIND ALPHA - THE ANGLE BETWEEN             */
/*   PHI AND THE EQUATOR OF THE EARTH           */
/************************************************/
    if ((Phi >= 1.5707963267) && (Phi <= 1.5707963269))
    {   Phi = 1.5707963267;
        ErrorList.AddError("CompareOrbit",
                              "Phi = 90 deg.  Phi adjusted for tan calculation",
                              0);
    }

    if (tan(Phi) != 0.0)
        CosineAlpha =   tan(RAInRad - (ThetaGInRad + LonInRadians)) / tan(Phi);
    else
    {   sprintf(buffer,"tan(Phi) = 0.0 forSatellite SSC: %d",
                    Sat.GetSSCNumber());
        ErrorList.AddError("CompareOrbit",
                              buffer,
                              1);
    }
```

```
/**********************************************/
/*   CONTINUE UNLESS CRITICAL ERROR           */
/**********************************************/
    if (ErrorList.CriticalError())
        return;


    if (CosineAlpha <= -1.0)
        CosineAlpha = -1.00;
    if (CosineAlpha >= 1.0)
        CosineAlpha = 1.00;

    Alpha = acos(CosineAlpha);



/**********************************************/
/*   FIND BETA - THE ANGLE BETWEEN            */
/*   PHI AND THE SATELLITE EPHEMERIS PATH     */
/**********************************************/
    if (Platform.GetLatitudeHemisphere() == 0)
    {   if ((Alpha + InclinInRad) <= (PI/2.0))
            Beta = Alpha + InclinInRad;
        else if ((Alpha + InclinInRad) <= PI)
            Beta = PI - Alpha - InclinInRad;
        else if ((Alpha + InclinInRad) <= (3.0*PI/2.0))
            Beta = Alpha + InclinInRad - PI;
        else if ((Alpha + InclinInRad) <= TWOPI)
            Beta = TWOPI - Alpha - InclinInRad;
        else
        {   ErrorList.AddError("CompareOrbit",
                                "Error computing Beta!!!",
                                1);
        }
    }
    else if (Platform.GetLatitudeHemisphere() == 1)
    {   if ((Alpha - InclinInRad) <= (-PI/2.0))
            Beta = PI + Alpha - InclinInRad;
        else if ((Alpha - InclinInRad) <= 0.0)
            Beta = InclinInRad - Alpha;
        else if ((Alpha - InclinInRad) <= (PI/2.0))
            Beta = Alpha - InclinInRad;
        else if ((Alpha - InclinInRad) <= PI)
            Beta = PI + InclinInRad - Alpha;
        else
        {   ErrorList.AddError("CompareOrbit",
                                "Error computing Beta!!!",
                                1);
        }
    }
    else
    {   ErrorList.AddError("CompareOrbit",
                            "Error computing Beta!!!",
                            1);
    }

/**********************************************/
/*   CONTINUE UNLESS CRITICAL ERROR           */
/**********************************************/
    if (ErrorList.CriticalError())
        return;
```

```
/************************************************/
/*   FIND D - THE MINIMUM SPHERICAL ANGLE       */
/*   BETWEEN THE SATELLITE EPHEMERIS PATH       */
/*   AND THE PLATFORM (MINIMUM DISTANCE)        */
/************************************************/
     SineD = sin(Beta) * sin(Phi);
     D = asin(SineD);

/*********************************************************************/
/*  HERE WE FIMD THE COMBINED ANGLE OF THE TRUE ANOMALY + ARGUMENT OF   */
/*  PERIGEE GIVEN THAT WE KNOW THE SIN AND COS OF THAT ANGLE...IE:      */
/*  GIVEN     sin(x) = y                                                 */
/*            cos(x) = z                                                 */
/*                                                                      */
/*  IF  sin-1(y) is positive or 0 AND cos-1(z) is positive or 0 THEN    */
/*     x = sin-1(y)                                                    */
/*  IF  sin-1(y) is positive or 0 AND cos-1(z) is negative THEN         */
/*     x = 180 deg - sin-1(y)                                          */
/*  IF  sin-1(y) is negative AND cos-1(z) is positive or 0 THEN         */
/*     x = 360 deg - sin-1(y)                                          */
/*  IF  sin-1(y) is negative AND cos-1(z) is negative THEN              */
/*         x = 180 deg + sin-1(y)                                          */
/*                                                                      */
/*  In Radians, 180 = PI, 360 = 2PI                                     */
/*********************************************************************/
    if (D >= 1.5707963267)
    {   D = 1.5707963266;
        ErrorList.AddError("CompareOrbit",
                           "D = 90 deg, D adjusted.",
                           0);
    }

    if ((Beta >= 1.5707963267) && (Beta <= 1.5707963269))
    {   Beta = 1.5707963266;
        ErrorList.AddError("CompareOrbit",
                           "Beta = 90 deg, Beta adjusted.",
                           0);
    }

    if (tan(Beta) != 0.0)
        SineOfVandW1 = tan(D)/tan(Beta);
    else
    {   sprintf(buffer,"tan(Beta) = 0.0 for Satellite SSC: %d",
                    Sat.GetSSCNumber());
        ErrorList.AddError("CompareOrbit",
                            buffer,
                            1);
    }

/************************************************/
/*   CONTINUE UNLESS CRITICAL ERROR             */
/************************************************/
    if (ErrorList.CriticalError())
        return;

    if (SineOfVandW1 >= 1.0)
        SineOfVandW1 = 1.0000000000000;

    VandW1 = asin(SineOfVandW1);

    if (cos(D) != 0.0)
        CosineOfVandW2 = cos(Phi)/cos(D);
    else
```

```
    {   sprintf(buffer,"cos(D) = 0.0 forSatellite SSC: %d",
                    Sat.GetSSCNumber());
        ErrorList.AddError("CompareOrbit",
                           buffer,
                           1);
    }
/**********************************************/
/*   CONTINUE UNLESS CRITICAL ERROR           */
/**********************************************/
    if (ErrorList.CriticalError())
        return;

    VandW2 = acos(CosineOfVandW2);

    if ((VandW1 >= 0.0) && (VandW2 >= 0.0))
        VandW = VandW1;
    else if ((VandW1 >= 0.0) && (VandW2 < 0.0))
        VandW = PI - VandW1;
    else if ((VandW1 < 0.0) && (VandW2 >= 0.0))
        VandW = TWOPI - VandW1;
    else if ((VandW1 < 0.0) && (VandW2 < 0.0))
        VandW = PI + VandW1;

    WInRad = Sat.GetArgumentOfPerigee() * DEGTORADIANS;
    TrueAnomalyInRad = VandW - WInRad;

/**************************************************/
/*   DERIVE SEMIMAJOR AXIS FROM THE SATELLITE     */
/*   MEAN MOTION.  SEMIMAJOR AXIS IS IN KILOMETERS */
/**************************************************/

    X = MMREVSPERDAY / Sat.GetMeanMotion();
    Y = 2.0/3.0;
    SemiMajorAxis =  pow(X,Y);

/**************************************************/
/*  FIND SATELLITE RADIUS FROM CENTER OF EARTH AT  */
/*  THE TRUE ANOMALY ANGLE FOUND PREVIOUSLY.       */
/**************************************************/
    Eccentricity = Sat.GetEccentricity();
    Numerator = SemiMajorAxis * (1.0 - pow(Eccentricity, 2.0));
    Denominator = 1.0 + Eccentricity * cos(TrueAnomalyInRad);
    SatRadius = Numerator/Denominator;

/**************************************************/
/*  FIND CRITICAL RADIUS FROM CENTER OF EARTH AT   */
/*  THE CLOSEST APPROACH. (CLOSEAST APPROACH WILL  */
/*  OCCUR AT THE TRUE ANOMALY ANGLE DERIVED ABOVE) */
/**************************************************/
    Altitude = Platform.GetAltitude();
    CriticalRadius = (EARTHRADIUS + Altitude )/ cos(D);

/**************************************************/
/*  COMPARE SATELLITE RADIUS TO THE CRITICAL       */
/*  RADIUS.  IF SATELLITE RADIUS IS BIGGER, THEN   */
/*  THE SATELLITE IS IN RANGE.                     */
/**************************************************/
    OrbitInView = 0;
    if (SatRadius >= CriticalRadius)
        OrbitInView = 1;

    return;
}
```

D.4 FindDisplacementAngleModules.cpp

```
/*****************************************************************************/
/*  MODULE NAME:    FindDisplacementAngleModules.cpp                       */
/*  AUTHOR:         Captain David Vloedman                                 */
/*  DATE CREATED:   3 January, 1999                                        */
/*                                                                         */
/*  PURPOSE:        This set of modules supports the Main Processor and are */
/*                  used to evaluate the error angle and the displacement   */
/*                  angle between the laser position vector in the REN frame*/
/*                  and the satellite position vector in the same frame.    */
/*                                                                         */
/*  COMPILER:       Borland C++ Builder3 Standard version                  */
/*                  This compiler should be used to compile and link.       */
/*                                                                         */
/*****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "TimeModules.h"
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisModules.h"
#include "SGP4SupportModules.h"
#include "FindDisplacementAngleModules.h"
#include "TargetSatellite.h"
#include "TargetPlatform.h"
#include "TargetLaser.h"
/**********************************/
/* C STANDARD LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/*****************************************************************************/
/***********************       FUCTIONS       ****************************/
/*****************************************************************************/

/*****************************************************************************/
/*  FUNCTION NAME:  FindDisplacementAngles                                 */
/*  AUTHOR:         Captain David Vloedman                                 */
/*  DATE CREATED:   January 3, 1999                                        */
/*                                                                         */
/*  PURPOSE:        This function will take satellite and platform data and */
/*                  willuse it to find the error angle and the displacement */
/*                  angle between the laser position vector in the REN frame*/
/*                  and the satellite position vector in the same frame.    */
/*                                                                         */
/*  INPUTS:         NAME:                   DEFINITION:                    */
```

```
/*                   Sat                      Holds all ephemeris information */
/*                                            for the Satellite being studied */
/*                   ABLPlatform              Holds all information about ABL */
/*                                            Platform position/disposition   */
/*                   JulianDate               The time to which the position  */
/*                                            of sat should be propagated to  */
/*                   ThetaGInRadians          The angle between the Greenwich */
/*                                            Meridian and the Vernal Equinox */
/*                                            at JulianDate.                  */
/*                   LazerAzimuthInDegrees    Lazer Azimuth at Laze Start time*/
/*                                            in Degrees                      */
/*                   LazerAzimuthDot          The rate of change of the Az    */
/*                                            in Degrees/Sec.                 */
/*                   LazerAzimuthDotDot       The rate of change of the rate  */
/*                                            of change of the Azimuth (Accel)*/
/*                                            in Degrees/Sec^2                */
/*                   LazerElevationInDegrees Lazer Elevation at Laze Start   */
/*                                            in Degrees                      */
/*                   LazerElevationDot        The rate of change of the El    */
/*                                            in Degrees/Sec.                 */
/*                   LazerElevationDotDot     The rate of change of the rate  */
/*                                            of change of the Elevat. (Accel)*/
/*                                            in Degrees/Sec^2                */
/*                   SatPositionErrorInMeters Holds the radius of the error   */
/*                                            spheroid that describes the     */
/*                                            area in which the satellite is  */
/*                                            known to exist (in meters).     */
/*                   PlatformPositionError...Holds the radius of the error    */
/*                                            spheroid that describes the     */
/*                                            area in which the platform is   */
/*                                            known to exist (in meters).     */
/*                   MissilePositionError... Holds the radius of the error    */
/*                                            spheroid that describes the     */
/*                                            area in which the missile is    */
/*                                            known to exist (in meters).     */
/*                   RangeToMissileInKilo... The Range to the missile (km)    */
/*                   OtherErrorAnglesInDeg    Holds any other error angles    */
/*                                            (in degrees) that may be a      */
/*                                            significant source of error.    */
/*                                            This should usually be set to   */
/*                                            zero (0.0) float.               */
/*  OUTPUTS:         NAME:                    DESCRIPTION:                    */
/*                   PlatformSatRENRhoR       The Radial Component of the     */
/*                                            position vector of the satellite*/
/*                                            wrt the platform in the REN     */
/*                                            coordinate frame.               */
/*                   PlatformSatRENRhoE       The East Component of the       */
/*                                            position vector of the satellite*/
/*                                            wrt the platform in the REN     */
/*                                            coordinate frame.               */
/*                   PlatformSatRENRhoN       The North Component of the      */
/*                                            position vector of the satellite*/
/*                                            wrt the platform in the REN     */
/*                                            coordinate frame.               */
/*                   PlatformSatRENRhoRDot    The Radial Component of the     */
/*                                            velocity vector of the satellite*/
/*                                            wrt the platform in the REN     */
/*                                            coordinate frame.               */
/*                   PlatformSatRENRhoEDot    The East Component of the       */
/*                                            velocity vector of the satellite*/
/*                                            wrt the platform in the REN     */
/*                                            coordinate frame.               */
/*                   PlatformSatRENRhoNDot    The North Component of the      */
```

```
/*                                          velocity vector of the satellite*/
/*                                          wrt the platform in the REN     */
/*                                          coordinate frame.               */
/*               PlatformSatRENRhoRDotDot The Radial Component of the     */
/*                                          accel vector of the satellite   */
/*                                          wrt the platform in the REN     */
/*                                          coordinate frame.               */
/*               PlatformSatRENRhoEDotDot The East Component of the       */
/*                                          accel vector of the satellite   */
/*                                          wrt the platform in the REN     */
/*                                          coordinate frame.               */
/*               PlatformSatRENRhoNDotDot The North Component of the      */
/*                                          accel vector of the satellite   */
/*                                          wrt the platform in the REN     */
/*                                          coordinate frame.               */
/*               LaserRENRhoR               The Radial unit direction of the*/
/*                                          lazer beam trajectory in the REN*/
/*                                          frame.                          */
/*               LaserRENRhoE               The East unit direction of the  */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame.                          */
/*               LaserRENRhoN               The North unit direction of the */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame.                          */
/*               LaserRENRhoRDot            The Radial unit velocity of the */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame in unit dirXradians/sec   */
/*               LaserRENRhoEDot            The East unit velocity of the   */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame in unit dirXradians/sec   */
/*               LaserRENRhoNDot            The North unit velocity of the  */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame in unit dirXradians/sec   */
/*               LaserRENRhoRDotDot         The Radial unit accel. of the   */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame in unit dirXradians/sec^2 */
/*               LaserRENRhoEDotDot         The East unit accel. of the     */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame in unit dirXradians/sec^2 */
/*               LaserRENRhoNDotDot         The North unit accel. of the    */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame in unit dirXradians/sec^2 */
/*               RangeInKilometers          Holds the range of the aircraft */
/*                                          to the satellite in kilometers. */
/*               ErrorAngleInRadians        The total error angle in radians*/
/*               SeparationAngle            The separation (in radians) of  */
/*                                          the LaserRENRho and             */
/*                                          PlatformSatRENRho vectors.      */
/*               SeparationAngleDot         The rate of change (in rad/sec) */
/*                                          of the separation of LaserRENRho*/
/*                                          PlatformSatRENRho vectors.      */
/*               SeparationAngleDotDot      The acceleration (in rad/sec^2) */
/*                                          of the separation of LaserRENRho*/
/*                                          and PlatformSatRENRho vectors.  */
/*               ErrorList                  The Errors which have occurred  */
/*                                                                          */
/*  COMPILER:    Borland C++ Builder3 Standard version                      */
/*               This compiler should be used to compile and link.          */
/*                                                                          */
/****************************************************************************/
void FindDisplacementAngles(struct Aircraft &Platform,
                             struct Satellite &Sat,
                             double &ThetaGInRad,
```

```
                                    double JulianDate,
                                    double LaserAzimuthInDegrees,
                                    double LaserAzimuthDot,
                                    double LaserAzimuthDotDot,
                                    double LaserElevationInDegrees,
                                    double LaserElevationDot,
                                    double LaserElevationDotDot,
                                    double SatPositionErrorInMeters,
                                    double PlatformPositionErrorInMeters,
                                    double MissilePositionErrorInMeters,
                                    double RangeToMissileInKilometers,
                                    double OtherErrorAngleInDeg,
                                    double &PlatformSatRENRhoR,
                                    double &PlatformSatRENRhoE,
                                    double &PlatformSatRENRhoN,
                                    double &PlatformSatRENRhoRDot,
                                    double &PlatformSatRENRhoEDot,
                                    double &PlatformSatRENRhoNDot,
                                    double &PlatformSatRENRhoRDotDot,
                                    double &PlatformSatRENRhoEDotDot,
                                    double &PlatformSatRENRhoNDotDot,
                                    double &LaserRENRhoR,
                                    double &LaserRENRhoE,
                                    double &LaserRENRhoN,
                                    double &LaserRENRhoRDot,
                                    double &LaserRENRhoEDot,
                                    double &LaserRENRhoNDot,
                                    double &LaserRENRhoRDotDot,
                                    double &LaserRENRhoEDotDot,
                                    double &LaserRENRhoNDotDot,
                                    double &RangeToSatInKilometers,
                                    double &ErrorAngleInRadians,
                                    double &SeparationAngle,
                                    double &SepAngleDot,
                                    double &SepAngleDotDot,
                                    ErrorStructure   &ErrorList)

{
/********************************/
/*  VARIABLE DECLARATIONS       */
/********************************/
    double SatECIRhoX;
    double *SatECIRhoXPtr = &SatECIRhoX;
    double SatECIRhoY;
    double *SatECIRhoYPtr = &SatECIRhoY;
    double SatECIRhoZ;
    double *SatECIRhoZPtr = &SatECIRhoZ;
    double SatECIRhoXDot;
    double *SatECIRhoXDotPtr = &SatECIRhoXDot;
    double SatECIRhoYDot;
    double *SatECIRhoYDotPtr = &SatECIRhoYDot;
    double SatECIRhoZDot;
    double *SatECIRhoZDotPtr = &SatECIRhoZDot;
    double SatECIRhoXDotDot;
    double *SatECIRhoXDotDotPtr = &SatECIRhoXDotDot;
    double SatECIRhoYDotDot;
    double *SatECIRhoYDotDotPtr = &SatECIRhoYDotDot;
    double SatECIRhoZDotDot;
    double *SatECIRhoZDotDotPtr = &SatECIRhoZDotDot;
    double SatRENRhoR;
    double *SatRENRhoRPtr = &SatRENRhoR;
    double SatRENRhoE;
```

```
double *SatRENRhoEPtr = &SatRENRhoE;
double SatRENRhoN;
double *SatRENRhoNPtr = &SatRENRhoN;
double SatRENRhoRDot;
double *SatRENRhoRDotPtr = &SatRENRhoRDot;
double SatRENRhoEDot;
double *SatRENRhoEDotPtr = &SatRENRhoEDot;
double SatRENRhoNDot;
double *SatRENRhoNDotPtr = &SatRENRhoNDot;
double SatRENRhoRDotDot;
double *SatRENRhoRDotDotPtr = &SatRENRhoRDotDot;
double SatRENRhoEDotDot;
double *SatRENRhoEDotDotPtr = &SatRENRhoEDotDot;
double SatRENRhoNDotDot;
double *SatRENRhoNDotDotPtr = &SatRENRhoNDotDot;
double PlatformECIRhoX;
double *PlatformECIRhoXPtr = &PlatformECIRhoX;
double PlatformECIRhoY;
double *PlatformECIRhoYPtr = &PlatformECIRhoY;
double PlatformECIRhoZ;
double *PlatformECIRhoZPtr = &PlatformECIRhoZ;
double PlatformECIRhoXDot;
double *PlatformECIRhoXDotPtr = &PlatformECIRhoXDot;
double PlatformECIRhoYDot;
double *PlatformECIRhoYDotPtr = &PlatformECIRhoYDot;
double PlatformECIRhoZDot;
double *PlatformECIRhoZDotPtr = &PlatformECIRhoZDot;
double PlatformECIRhoXDotDot;
double *PlatformECIRhoXDotDotPtr = &PlatformECIRhoXDotDot;
double PlatformECIRhoYDotDot;
double *PlatformECIRhoYDotDotPtr = &PlatformECIRhoYDotDot;
double PlatformECIRhoZDotDot;
double *PlatformECIRhoZDotDotPtr = &PlatformECIRhoZDotDot;
double PlatformRENRhoR;
double *PlatformRENRhoRPtr = &PlatformRENRhoR;
double PlatformRENRhoE;
double *PlatformRENRhoEPtr = &PlatformRENRhoE;
double PlatformRENRhoN;
double *PlatformRENRhoNPtr = &PlatformRENRhoN;
double PlatformRENRhoRDot;
double *PlatformRENRhoRDotPtr = &PlatformRENRhoRDot;
double PlatformRENRhoEDot;
double *PlatformRENRhoEDotPtr = &PlatformRENRhoEDot;
double PlatformRENRhoNDot;
double *PlatformRENRhoNDotPtr = &PlatformRENRhoNDot;
double PlatformRENRhoRDotDot;
double *PlatformRENRhoRDotDotPtr = &PlatformRENRhoRDotDot;
double PlatformRENRhoEDotDot;
double *PlatformRENRhoEDotDotPtr = &PlatformRENRhoEDotDot;
double PlatformRENRhoNDotDot;
double *PlatformRENRhoNDotDotPtr = &PlatformRENRhoNDotDot;
double  ECItoRENMatrix11;
double *ECItoRENMatrix11Ptr = &ECItoRENMatrix11;
double  ECItoRENMatrix12;
double *ECItoRENMatrix12Ptr = &ECItoRENMatrix12;
double  ECItoRENMatrix13;
double *ECItoRENMatrix13Ptr = &ECItoRENMatrix13;
double  ECItoRENMatrix21;
double *ECItoRENMatrix21Ptr = &ECItoRENMatrix21;
double  ECItoRENMatrix22;
double *ECItoRENMatrix22Ptr = &ECItoRENMatrix22;
double  ECItoRENMatrix23;
double *ECItoRENMatrix23Ptr = &ECItoRENMatrix23;
```

```
    double  ECItoRENMatrix31;
    double *ECItoRENMatrix31Ptr = &ECItoRENMatrix31;
    double  ECItoRENMatrix32;
    double *ECItoRENMatrix32Ptr = &ECItoRENMatrix32;
    double  ECItoRENMatrix33;
    double *ECItoRENMatrix33Ptr = &ECItoRENMatrix33;

/***************************************************/
/*   FIND THE PLATFORM POSITION, VELOCITY, AND     */
/*   ACCELERATION IN BOTH THE ECI AND REN          */
/*   COORDINATE FRAMES.  AFTER CONVERSION TO THE   */
/*   REN FRAME, ALSO RETURN THE ECI TO REN CON-    */
/*   VERSION MATRIX TO USE IN OTHER ROTATIONS.     */
/***************************************************/
    TargetPlatform(Platform,
                   ThetaGInRad,
                   JulianDate,
                   *PlatformECIRhoXPtr,
                   *PlatformECIRhoYPtr,
                   *PlatformECIRhoZPtr,
                   *PlatformECIRhoXDotPtr,
                   *PlatformECIRhoYDotPtr,
                   *PlatformECIRhoZDotPtr,
                   *PlatformECIRhoXDotDotPtr,
                   *PlatformECIRhoYDotDotPtr,
                   *PlatformECIRhoZDotDotPtr,
                   *PlatformRENRhoRPtr,
                   *PlatformRENRhoEPtr,
                   *PlatformRENRhoNPtr,
                   *PlatformRENRhoRDotPtr,
                   *PlatformRENRhoEDotPtr,
                   *PlatformRENRhoNDotPtr,
                   *PlatformRENRhoRDotDotPtr,
                   *PlatformRENRhoEDotDotPtr,
                   *PlatformRENRhoNDotDotPtr,
                   *ECItoRENMatrix11Ptr,       /**********************/
                   *ECItoRENMatrix12Ptr,       /*  ECI TO REN MATRIX */
                   *ECItoRENMatrix13Ptr,       /*  USED TO CONVERT   */
                   *ECItoRENMatrix21Ptr,       /*  FROM ECI TO REN   */
                   *ECItoRENMatrix22Ptr,       /*  COORDINATES.      */
                   *ECItoRENMatrix23Ptr,       /**********************/
                   *ECItoRENMatrix31Ptr,
                   *ECItoRENMatrix32Ptr,
                   *ECItoRENMatrix33Ptr,
                   ErrorList);

/***************************************************/
/*  FIND THE SATELLITE POSITION, VELOCITY AND      */
/*  ACCELERATION IN THE ECI FRAME, THEN USE THE    */
/*  ECI TO REN CON MATRIX TO FIND THE REN VERSION. */
/***************************************************/
    TargetSatellite(Sat,
                    JulianDate,
                    ECItoRENMatrix11,
                    ECItoRENMatrix12,
                    ECItoRENMatrix13,
                    ECItoRENMatrix21,
                    ECItoRENMatrix22,
                    ECItoRENMatrix23,
                    ECItoRENMatrix31,
                    ECItoRENMatrix32,
                    ECItoRENMatrix33,
                    *SatECIRhoXPtr,
```

```
                        *SatECIRhoYPtr,
                        *SatECIRhoZPtr,
                        *SatECIRhoXDotPtr,
                        *SatECIRhoYDotPtr,
                        *SatECIRhoZDotPtr,
                        *SatECIRhoXDotDotPtr,
                        *SatECIRhoYDotDotPtr,
                        *SatECIRhoZDotDotPtr,
                        *SatRENRhoRPtr,
                        *SatRENRhoEPtr,
                        *SatRENRhoNPtr,
                        *SatRENRhoRDotPtr,
                        *SatRENRhoEDotPtr,
                        *SatRENRhoNDotPtr,
                        *SatRENRhoRDotDotPtr,
                        *SatRENRhoEDotDotPtr,
                        *SatRENRhoNDotDotPtr,
                        ErrorList);

/***************************************************/
/*  FIND POSITION, VELOCITY AND ACCELERATION       */
/*  VALUES OF VECTOR GOING FROM PLATFORM TO        */
/*  SATELLITE IN PLATFORM-CENTERED REN FRAME       */
/***************************************************/

/*****************/
/* POSITION      */
/*****************/
    PlatformSatRENRhoR = SatRENRhoR - PlatformRENRhoR;
    PlatformSatRENRhoE = SatRENRhoE - PlatformRENRhoE;
    PlatformSatRENRhoN = SatRENRhoN - PlatformRENRhoN;

/*****************/
/* VELOCITY      */
/*****************/
    PlatformSatRENRhoRDot = SatRENRhoRDot - PlatformRENRhoRDot;
    PlatformSatRENRhoEDot = SatRENRhoEDot - PlatformRENRhoEDot;
    PlatformSatRENRhoNDot = SatRENRhoNDot - PlatformRENRhoNDot;

/*****************/
/* ACCELERATION  */
/*****************/
    PlatformSatRENRhoRDotDot = SatRENRhoRDotDot - PlatformRENRhoRDotDot;
    PlatformSatRENRhoEDotDot = SatRENRhoEDotDot - PlatformRENRhoEDotDot;
    PlatformSatRENRhoNDotDot = SatRENRhoNDotDot - PlatformRENRhoNDotDot;


/************************************/
/*  FIND RANGE TO SATELLITE         */
/************************************/
    RangeToSatInKilometers = sqrt(pow(PlatformSatRENRhoR,2) +
                                  pow(PlatformSatRENRhoE,2) +
                                  pow(PlatformSatRENRhoN,2));

/***************************************************/
/*  FIND THE ERROR HALF-ANGLE ASSOCIATED WITH THE  */
/*  UNCERTAINTY IN THE SATELLITES POSITION         */
/***************************************************/
    FindErrorAngle(RangeToSatInKilometers,
                   SatPositionErrorInMeters,
                   PlatformPositionErrorInMeters,
                   MissilePositionErrorInMeters,
                   RangeToMissileInKilometers,
```

```
                    OtherErrorAngleInDeg,
                    ErrorAngleInRadians,
                    ErrorList);


/****************************************************/
/*   FIND THE VECTOR IN THE REN FRAME ASSOCIATED    */
/*   THE CURRENT AZIMUTH AND ELEVATION.  THE        */
/*   VECTOR RETURNED (LaserRENRho) IS THE UNIT      */
/*   DIRECTION VECTOR POINTING IN THE SAME DIR      */
/*   AS THE AZIMUTH AND ELEVATION.                  */
/****************************************************/
TargetLaser(LaserAzimuthInDegrees,
            LaserElevationInDegrees,
            LaserAzimuthDot,
            LaserElevationDot,
            LaserAzimuthDotDot,
            LaserElevationDotDot,
            LaserRENRhoR,
            LaserRENRhoE,
            LaserRENRhoN,
            LaserRENRhoRDot,
            LaserRENRhoEDot,
            LaserRENRhoNDot,
            LaserRENRhoRDotDot,
            LaserRENRhoEDotDot,
            LaserRENRhoNDotDot,
            ErrorList);

/****************************************************/
/*  FIND THE ANGLE THAT SEPARATES THE SATELLITE     */
/*  POSITION VECTOR AND THE LASER TURRET UNIT       */
/*  DIRECTION VECTOR.                               */
/****************************************************/
    FindSeparationAngle(LaserRENRhoR,
                        LaserRENRhoE,
                        LaserRENRhoN,
                        LaserRENRhoRDot,
                        LaserRENRhoEDot,
                        LaserRENRhoNDot,
                        LaserRENRhoRDotDot,
                        LaserRENRhoEDotDot,
                        LaserRENRhoNDotDot,
                        PlatformSatRENRhoR,
                        PlatformSatRENRhoE,
                        PlatformSatRENRhoN,
                        PlatformSatRENRhoRDot,
                        PlatformSatRENRhoEDot,
                        PlatformSatRENRhoNDot,
                        PlatformSatRENRhoRDotDot,
                        PlatformSatRENRhoEDotDot,
                        PlatformSatRENRhoNDotDot,
                        SeparationAngle,
                        SepAngleDot,
                        SepAngleDotDot,
                        ErrorList);

    return;

}
```

```
/**************************************************************************/
/*  FUNCTION NAME:  FindErrorAngle                                        */
/*  AUTHOR:         Captain David Vloedman                                */
/*  DATE CREATED:   January 3, 1999                                       */
/*                                                                        */
/*  PURPOSE:        This function will take the range to satellite and the */
/*                  satellite position error and fiond the appropriate error*/
/*                  error angle.                                          */
/*                                                                        */
/*  INPUTS:         NAME:                   DEFINITION:                   */
/*                  Range                   Holds the range of the aircraft */
/*                                          to the satellite in kilometers. */
/*                  SatPositionErrorInMeters Holds the radius of the error  */
/*                                          spheroid that describes the    */
/*                                          area in which the satellite is */
/*                                          known to exist (in meters).    */
/*                  PlatformPositionError...Holds the radius of the error  */
/*                                          spheroid that describes the    */
/*                                          area in which the platform is  */
/*                                          known to exist (in meters).    */
/*                  MissilePositionError... Holds the radius of the error  */
/*                                          spheroid that describes the    */
/*                                          area in which the missile is   */
/*                                          known to exist (in meters).    */
/*                  RangeToMissileInKilo... The Range to the missile (km)  */
/*                  OtherErrorAnglesInDeg   Holds any other error angles   */
/*                                          (in degrees) that may be a     */
/*                                          significant source of error.   */
/*                                          This should usually be set to  */
/*                                          zero (0.0) float.              */
/*  OUTPUTS:        NAME:                   DESCRIPTION:                   */
/*                  ErrorAngleInRadians     The total error angle in radians*/
/*                  ErrorList               The Errors which have occurred */
/*                                                                        */
/*  COMPILER:       Borland C++ Builder3 Standard version                 */
/*                  This compiler should be used to compile and link.     */
/*                                                                        */
/**************************************************************************/
void FindErrorAngle(double RangeToSatInKilometers,
                    double SatPositionErrorInMeters,
                    double PlatformPositionErrorInMeters,
                    double MissilePositionErrorInMeters,
                    double RangeToMissileInKilometers,
                    double OtherErrorAnglesInDeg,
                    double &ErrorAngleInRadians,
                    ErrorStructure   &ErrorList)
{

    double  SatErrorAngle;
    double  PlatformErrorAngle;
    double  MissileErrorAngle;
    double  RangeToSatInMeters;
    double  RangeToMissileInMeters;
    double  DisplacementAtSat;
    double  MissileToSatInMeters;
    char    buffer[MAXMESSAGELENGTH] = " ";

/**************************************************/
/*   ERROR CHECK EACH PARAMETER                   */
/**************************************************/
    if (RangeToSatInKilometers <= 0.0)
    {   sprintf(buffer,"Range cannot be zero or negative. Range = %d",
                    RangeToSatInKilometers);
```

```
        ErrorList.AddError("FindErrorAngle",
                           buffer,
                           1);
    }
    if (RangeToMissileInKilometers <= 0.0)
    {   sprintf(buffer,"Range to missile cannot be less than 0. Range = %d",
                    RangeToMissileInKilometers);
        ErrorList.AddError("FindErrorAngle",
                           buffer,
                           1);
    }
    if (SatPositionErrorInMeters < 0.0)
    {   sprintf(buffer,"Position Error cannot be negative. Pos Error = %d",
                    SatPositionErrorInMeters);
        ErrorList.AddError("FindErrorAngle",
                           buffer,
                           1);
    }
    if (PlatformPositionErrorInMeters < 0.0)
    {   sprintf(buffer,"Position Error cannot be negative. Pos Error = %d",
                    PlatformPositionErrorInMeters);
        ErrorList.AddError("FindErrorAngle",
                           buffer,
                           1);
    }
    if (MissilePositionErrorInMeters < 0.0)
    {   sprintf(buffer,"Position Error cannot be negative. Pos Error = %d",
                    MissilePositionErrorInMeters);
        ErrorList.AddError("FindErrorAngle",
                           buffer,
                           1);
    }
    if (OtherErrorAnglesInDeg < 0.0)
    {   sprintf(buffer,"Error Angles cannot be negative. Other Angles = %d",
                    OtherErrorAnglesInDeg);
        ErrorList.AddError("FindErrorAngle",
                           buffer,
                           1);
    }
    if (OtherErrorAnglesInDeg > 90.0)
    {   sprintf(buffer,"Error Angle is too big to work with: Angle= %d",
                    OtherErrorAnglesInDeg);
        ErrorList.AddError("FindErrorAngle",
                           buffer,
                           1);
    }

/**************************************************/
/*   INITIALIZE OUTPUT VARIABLES                  */
/**************************************************/
    ErrorAngleInRadians = 0.0;

/**************************************************/
/*   BEGIN CALCULATIONS UNLESS CRITICAL ERROR     */
/**************************************************/
    if (ErrorList.CriticalError())
        return;

/**************************************************/
/*   FIND ERROR DUE TO SATELLITE POSITION ERROR */
/**************************************************/
    RangeToSatInMeters = RangeToSatInKilometers * 1000;
    SatErrorAngle = atan(SatPositionErrorInMeters/RangeToSatInMeters);
```

```
/**************************************************/
/*   FIND ERROR DUE TO PLATFORM POSITION ERROR   */
/**************************************************/
    RangeToMissileInMeters = RangeToMissileInKilometers * 1000;
    MissileToSatInMeters = RangeToSatInMeters - RangeToMissileInMeters;
    DisplacementAtSat = MissileToSatInMeters *
                        PlatformPositionErrorInMeters /
                        RangeToMissileInMeters;
    PlatformErrorAngle = atan(DisplacementAtSat / RangeToSatInMeters);

/**************************************************/
/*   FIND ERROR DUE TO MISSILE POSITION ERROR    */
/**************************************************/
    MissileErrorAngle = atan(MissilePositionErrorInMeters /
                             RangeToMissileInMeters);

/**************************************************/
/*   FIND ERROR DUE TO ALL ERRORS COMBINED       */
/**************************************************/
     ErrorAngleInRadians = sqrt(pow(SatErrorAngle, 2) +
                                pow(PlatformErrorAngle, 2) +
                                pow(MissileErrorAngle, 2) +
                                pow(OtherErrorAnglesInDeg * DEGTORADIANS, 2));


    return;

}


/******************************************************************************/
/*  FUNCTION NAME:  FindSeparationAngle                                       */
/*  AUTHOR:         Captain David Vloedman                                    */
/*  DATE CREATED:   January 3, 1999                                           */
/*                                                                            */
/*  PURPOSE:        This routine finds the angle separating the satellite    */
/*                  position vector and the laser turret unit direction      */
/*                  vector in the REN coordinate frame, as well as the rate */
/*                  of change and the acceleration of that separation.       */
/*                                                                            */
/*  INPUTS:         NAME:                   DEFINITION:                       */
/*                  LaserRENRhoR            The Radial unit direction of the*/
/*                                          lazer beam trajectory in the REN*/
/*                                          frame.                            */
/*                  LaserRENRhoE            The East unit direction of the   */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame.                            */
/*                  LaserRENRhoN            The North unit direction of the */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame.                            */
/*                  LaserRENRhoRDot         The Radial unit velocity of the */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame in unit dir*radians/sec    */
/*                  LaserRENRhoEDot         The East unit velocity of the    */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame in unit dir*radians/sec    */
/*                  LaserRENRhoNDot         The North unit velocity of the  */
/*                                          lazer beam trajectory in the REN*/
/*                                          frame in unit dir*radians/sec    */
/*                  LaserRENRhoRDotDot      The Radial unit accel. of the    */
/*                                          lazer beam trajectory in the REN*/
```

```
/*                                        frame in unit dir*radians/sec^2 */
/*               LaserRENRhoEDotDot       The East unit accel. of the     */
/*                                        lazer beam trajectory in the REN*/
/*                                        frame in unit dir*radians/sec^2 */
/*               LaserRENRhoNDotDot       The North unit accel. of the    */
/*                                        lazer beam trajectory in the REN*/
/*                                        frame in unit dir*radians/sec^2 */
/*               SatRENRhoR               The Radial Component of the     */
/*                                        position vector of the satellite*/
/*                                        wrt the platform in the REN     */
/*                                        coordinate frame.               */
/*               SatRENRhoE               The East Component of the       */
/*                                        position vector of the satellite*/
/*                                        wrt the platform in the REN     */
/*                                        coordinate frame.               */
/*               SatRENRhoN               The North Component of the      */
/*                                        position vector of the satellite*/
/*                                        wrt the platform in the REN     */
/*                                        coordinate frame.               */
/*               SatRENRhoRDot            The Radial Component of the     */
/*                                        velocity vector of the satellite*/
/*                                        wrt the platform in the REN     */
/*                                        coordinate frame.               */
/*               SatRENRhoEDot            The East Component of the       */
/*                                        velocity vector of the satellite*/
/*                                        wrt the platform in the REN     */
/*                                        coordinate frame.               */
/*               SatRENRhoNDot            The North Component of the      */
/*                                        velocity vector of the satellite*/
/*                                        wrt the platform in the REN     */
/*                                        coordinate frame.               */
/*               SatRENRhoRDotDot         The Radial Component of the     */
/*                                        accel vector of the satellite   */
/*                                        wrt the platform in the REN     */
/*                                        coordinate frame.               */
/*               SatRENRhoEDotDot         The East Component of the       */
/*                                        accel vector of the satellite   */
/*                                        wrt the platform in the REN     */
/*                                        coordinate frame.               */
/*               SatRENRhoNDotDot         The North Component of the      */
/*                                        accel vector of the satellite   */
/*                                        wrt the platform in the REN     */
/*                                        coordinate frame.               */
/*                                                                        */
/*  OUTPUTS:     NAME:                    DESCRIPTION:                    */
/*               SeparationAngle          The separation (in radians) of  */
/*                                        the LaserRENRho and             */
/*                                        PlatformSatRENRho vectors.      */
/*               SeparationAngleDot       The rate of change (in rad/sec) */
/*                                        of the separation of LaserRENRho*/
/*                                        PlatformSatRENRho vectors.      */
/*               SeparationAngleDotDot    The acceleration (in rad/sec^2) */
/*                                        of the separation of LaserRENRho*/
/*                                        and PlatformSatRENRho vectors.  */
/*               ErrorList                The Errors which have occurred  */
/*                                                                        */
/*  COMPILER:    Borland C++ Builder3 Standard version                    */
/*               This compiler should be used to compile and link.        */
/*                                                                        */
/**************************************************************************/
void FindSeparationAngle(double LaserRENRhoR,
                         double LaserRENRhoE,
                         double LaserRENRhoN,
```

```
                          double LaserRENRhoRDot,
                          double LaserRENRhoEDot,
                          double LaserRENRhoNDot,
                          double LaserRENRhoRDotDot,
                          double LaserRENRhoEDotDot,
                          double LaserRENRhoNDotDot,
                          double SatRENRhoR,
                          double SatRENRhoE,
                          double SatRENRhoN,
                          double SatRENRhoRDot,
                          double SatRENRhoEDot,
                          double SatRENRhoNDot,
                          double SatRENRhoRDotDot,
                          double SatRENRhoEDotDot,
                          double SatRENRhoNDotDot,
                          double &SeparationAngleInRadians,
                          double &SepAngleDot,
                          double &SepAngleDotDot,
                          ErrorStructure   &ErrorList)
{
    double  MagnitudeSatREN;
    double  CosineSepAngle;
    double  PLDivP;
    double  PLDotDivP;
    double  PDotLDivP;
    double  PPDotDivP;
    double  PPDotDotDivP;
    double  PDotLDotDivP;
    double  PLDotDotDivP;
    double  PDotDotLDivP;
    double  PDotPDotDivP;
    double  U;
    double  V;
    double  dU;
    double  dV;


/*****************************************************/
/*  first, FIND MAGNITUDE OF RHO TO SATELLITE        */
/*****************************************************/
    MagnitudeSatREN = sqrt(pow(SatRENRhoR,2) +
                           pow(SatRENRhoE,2) +
                           pow(SatRENRhoN,2));

/*****************************************************/
/*  NEXT FIND THE SEPARATION ANGLE THAT DEFINES THE  */
/*  ANGLE BETWEEN THE SATELLITE VECTOR AND THE LASER */
/*  TURRET VECTOR. SEE THE THESIS BREAKDOWN OF THE   */
/*  FOLLOWING FORMULA TO UNDERSTAND THE DERIVATION.  */
/*****************************************************/
    CosineSepAngle = LaserRENRhoR * SatRENRhoR / MagnitudeSatREN +
                     LaserRENRhoE * SatRENRhoE / MagnitudeSatREN +
                     LaserRENRhoN * SatRENRhoN / MagnitudeSatREN;

    SeparationAngleInRadians = acos(CosineSepAngle);

/*****************************************************/
/*  NEXT FIND THE VELOCITY OR RATE OF CHANGE OF THE  */
/*  SEPARATION ANGLE THAT DEFINES THE ANGLE BETWEEN  */
/*  THE SATELLITE VECTOR AND THE LASER TURRET VECTOR. */
/*  SEE THE THESIS BREAKDOWN OF THE FOLLOWING FORMULA */
/*  TO UNDERSTAND THE DERIVATION.                    */
/*  IN THE FOLLOWING FORMULAS, SHORT NAMES HAVE BEEN  */
```

```
/* USED TO SUBSTITUTE FOR LONG NAMES:                   */
/* P        = SatRENRho   -> VECTOR FROM TURRET TO SAT */
/* PDot     = SatRENRhoDot -> RATE OF CHANGE OF ABOVE */
/* PDotDot = SatRENRhoDotDot -> ACCELERATION OF ...   */
/* L        = LaserRENRho -> UNIT DIR. OF LASER VECTOR*/
/* LDot     = LaserRENRhoDot -> VELOCITY OF """       */
/* LDotDot = LaserRENRhoDotDot -> ACCELERATION OF """*/
/*****************************************************/
    PLDivP =(SatRENRhoR * LaserRENRhoR +
             SatRENRhoE * LaserRENRhoE +
             SatRENRhoN * LaserRENRhoN)/
             MagnitudeSatREN;

    PLDotDivP =(SatRENRhoR * LaserRENRhoRDot +
                SatRENRhoE * LaserRENRhoEDot +
                SatRENRhoN * LaserRENRhoNDot)/
                MagnitudeSatREN;

    PDotLDivP =(SatRENRhoRDot * LaserRENRhoR +
                SatRENRhoEDot * LaserRENRhoE +
                SatRENRhoNDot * LaserRENRhoN)/
                MagnitudeSatREN;

    PPDotDivP =(SatRENRhoR * SatRENRhoRDot +
                SatRENRhoE * SatRENRhoEDot +
                SatRENRhoN * SatRENRhoNDot)/
                MagnitudeSatREN;


    SepAngleDot = -pow(1-pow(PLDivP,2),-0.5) *
                  (PLDotDivP +
                   PDotLDivP -
                  (PLDivP/MagnitudeSatREN) *
                   PPDotDivP);

/*****************************************************/
/* FINALLY, FIND THE ACCELERATION OR RATE OF CHANGE  */
/* OF THE VELOCITY OF THE SEPARATION ANGLE THAT      */
/* DEFINES THE ANGLE BETWEEN THE SATELLITE VECTOR AND*/
/* THE LASER TURRET VECTOR.  SEE THE THESIS BREAKDOWN*/
/* OF THE FOLLOWING FORMULA TO UNDERSTAND THE        */
/* DERIVATION.                                       */
/* P        = SatRENRho   -> VECTOR FROM TURRET TO SAT */
/* PDot     = SatRENRhoDot -> RATE OF CHANGE OF ABOVE */
/* PDotDot = SatRENRhoDotDot -> ACCELERATION OF ...   */
/* L        = LaserRENRho -> UNIT DIR. OF LASER VECTOR*/
/* LDot     = LaserRENRhoDot -> VELOCITY OF """       */
/* LDotDot = LaserRENRhoDotDot -> ACCELERATION OF """*/
/*****************************************************/
    PLDotDotDivP =(SatRENRhoR * LaserRENRhoRDotDot +
                   SatRENRhoE * LaserRENRhoEDotDot +
                   SatRENRhoN * LaserRENRhoNDotDot)/
                   MagnitudeSatREN;

    PDotLDotDivP =(SatRENRhoRDot * LaserRENRhoRDot +
                   SatRENRhoEDot * LaserRENRhoEDot +
                   SatRENRhoNDot * LaserRENRhoNDot)/
                   MagnitudeSatREN;

    PDotDotLDivP =(SatRENRhoRDotDot * LaserRENRhoR +
                   SatRENRhoEDotDot * LaserRENRhoE +
                   SatRENRhoNDotDot * LaserRENRhoN)/
                   MagnitudeSatREN;
```

```
    PPDotDotDivP =(SatRENRhoR * SatRENRhoRDotDot +
                   SatRENRhoE * SatRENRhoEDotDot +
                   SatRENRhoN * SatRENRhoNDotDot)/
                   MagnitudeSatREN;

    PDotPDotDivP =(SatRENRhoRDot * SatRENRhoRDot +
                   SatRENRhoEDot * SatRENRhoEDot +
                   SatRENRhoNDot * SatRENRhoNDot)/
                   MagnitudeSatREN;

    U = -pow((1-pow(PLDivP,2)),-0.5);

    dU = -pow((1-pow(PLDivP,2)),-1.5) *
          (PLDotDivP +
           PDotLDivP -
           (PLDivP/MagnitudeSatREN) *
            PPDotDivP) *
           PLDivP;

    V = PLDotDivP +
        PDotLDivP -
        (PLDivP/MagnitudeSatREN) *
        PPDotDivP;

   dV = (PLDotDotDivP +
         PDotLDotDivP -
         (PLDotDivP/MagnitudeSatREN) *
         PPDotDivP) +

        (PDotLDotDivP +
         PDotDotLDivP -
         (PDotLDivP/MagnitudeSatREN) *
         PPDotDivP) +

        (PPDotDivP *
         (2.0 * (PLDivP/MagnitudeSatREN) *
                (PPDotDivP/MagnitudeSatREN) -
         (PLDotDivP/MagnitudeSatREN +
          PDotLDivP/MagnitudeSatREN))) -

        (PLDivP/MagnitudeSatREN *
         (PPDotDotDivP +
          PDotPDotDivP -
          (PPDotDivP/MagnitudeSatREN) *
          PPDotDivP));

    SepAngleDotDot = U*dV + V*dU;

    return;
}
```

D.5 PAMainProcessor.cpp

```
/*****************************************************************************/
/*  MODULE NAME:     PAMainProcessor.cpp                                     */
/*  AUTHOR:          Captain David Vloedman                                  */
/*  DATE CREATED:    January 15, 1998                                        */
/*                                                                           */
/*  PURPOSE:         This module is the model of the Airborne Laser          */
/*                   Predictive Avoidance Processor which may be used to     */
/*                   determine whether or not a given Laser trajectory will  */
/*                   intersect with any of a list of satellites fed to it.   */
/*                                                                           */
/*  COMPILER:        Borland C++ Builder3 Standard version                   */
/*                   This compiler should be used to compile and link.       */
/*                                                                           */
/*****************************************************************************/
/*********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/*********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/*********************************/
/* USER-BUILT LIBRARIES          */
/*********************************/
#include "TimeModules.h"
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisModules.h"
#include "PAMainProcessor.h"
#include "SGP4SupportModules.h"
#include "FindDisplacementAngleModules.h"
#include "TargetSatellite.h"
#include "TargetPlatform.h"
#include "TargetLaser.h"
#include "ProcessSatellite.h"
/*********************************/
/* C STANDARD LIBRARIES          */
/*********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/*****************************************************************************/
/***********************        FUCTIONS        *****************************/
/*****************************************************************************/


/*****************************************************************************/
/*  FUNCTION NAME:   PAMainProcessor                                         */
/*  AUTHOR:          Captain David Vloedman                                  */
/*  DATE CREATED:    January 15, 1998                                        */
/*                                                                           */
/*  PURPOSE:         This procedure will read in an input file of Two Line   */
/*                   Element (TLE) sets and perform an analysis to determine */
/*                   whether or not satellites will be intercepted by the    */
```

```
/*                 path of the airborne platform laser.                          */
/*                                                                               */
/*  INPUTS:        NAME:                  DEFINITION:                            */
/*                 InFileName             Holds name of the satellite file*/
/*                 OutFileName            File that holds the sats that   */
/*                                        are forecasted by the software  */
/*                                        to be intercepted bt the laser. */
/*                 ClosestApproachFileName File that holds the sats that   */
/*                                        are forecasted by the software  */
/*                                        to be close to the laser. These */
/*                                        are not necessarily intersected.*/
/*                 ABLPlatform            Holds all information about ABL */
/*                                        Platform position/disposition   */
/*                 ReferenceHour          This holds the value of Theta G */
/*                                        at RefModJulianDate.  The angle */
/*                                        of Theta G is given in hours,   */
/*                                        minutes, and seconds instead of */
/*                                        degrees, where 24 hrs = 360 deg */
/*                 ReferenceMinute        Holds the minutes of Theta G at */
/*                                        RefModJulianDate.               */
/*                 ReferenceSecond        Holds the seconds of Theta G at */
/*                                        RefModJulianDate.               */
/*                 RefModJulianDate       This is the reference date when */
/*                                        an actual observation of the    */
/*                                        true value of theta G was made. */
/*                 CalcYear               Holds the current calender year */
/*                 Calcmonth              Holds the Calender month(1 - 12)*/
/*                 CalcDay                Holds calender day              */
/*                 CalcHour               Holds the calender hour         */
/*                 CalcMinute             Holds the calender minute       */
/*                 CalcSecond             Holds the calender second       */
/*                 LazeDuration           The amount of time for which the*/
/*                                        laser will be on.  This is      */
/*                                        To determine how much time in   */
/*                                        seconds the forecast will last. */
/*                 LazerAzimuthInDegrees  Lazer Azimuth at Laze Start time*/
/*                                        in Degrees                      */
/*                 LazerAzimuthDot        The rate of change of the Az    */
/*                                        in Degrees/Sec.                 */
/*                 LazerAzimuthDotDot     The rate of change of the rate  */
/*                                        of change of the Azimuth (Accel)*/
/*                                        in Degrees/Sec^2                */
/*                 LazerElevationInDegrees Lazer Elevation at Laze Start   */
/*                                        in Degrees                      */
/*                 LazerElevationDot      The rate of change of the El    */
/*                                        in Degrees/Sec.                 */
/*                 LazerElevationDotDot   The rate of change of the rate  */
/*                                        of change of the Elevat. (Accel)*/
/*                                        in Degrees/Sec^2                */
/*                 SatPositionErrorInMeters Holds the radius of the error  */
/*                                        spheroid that describes the     */
/*                                        area in which the satellite is  */
/*                                        known to exist (in meters).     */
/*                 PlatformPositionError...Holds the radius of the error   */
/*                                        spheroid that describes the     */
/*                                        area in which the platform is   */
/*                                        known to exist (in meters).     */
/*                 MissilePositionError... Holds the radius of the error   */
/*                                        spheroid that describes the     */
/*                                        area in which the missile is    */
/*                                        known to exist (in meters).     */
/*                 RangeToMissileInKilo... The Range to the missile (km)   */
/*                 OtherErrorAnglesInDeg  Holds any other error angles    */
```

```
/*                                              (in degrees) that may be a       */
/*                                              significant source of error.     */
/*                                              This should usually be set to    */
/*                                              zero (0.0) float.                */
/*                 ThetaGInRadians              The angle between the Greenwich */
/*                                              Meridian and the Vernal Equinox */
/*                                              at JulianDate.                   */
/*                                                                               */
/*  OUTPUTS:       NAME:                        DESCRIPTION:                     */
/*                 InFileLength                 The total number of satellites   */
/*                                              that have been evaluated in the */
/*                                              InFile                           */
/*                 OutFileLength                The total number of satellites   */
/*                                              that are intersected by platform*/
/*                                              and have been put in the outfile*/
/*                 ClosestApproachFileLength The total number of satellites*/
/*                                              that come close to the laser     */
/*                                              and have been put in the         */
/*                                              closest approach file.           */
/*                 ErrorList                    Errors that have occured         */
/*                                                                               */
/*                 THE FINAL OUTPUT IS THE ACTUAL OUTFILE ITSELF WHICH IS        */
/*                 WRITTEN DIRECTLY TO DISK SO IT CAN BE ACCESSED BY             */
/*                 OTHER SOFTWARE, IF NEEDED.                                    */
/*                                                                               */
/*  COMPILER:      Borland C++ Builder3 Standard version                         */
/*                 This compiler should be used to compile and link.             */
/*                                                                               */
/*******************************************************************************/
PAMainProcessor(char     InFileName[MAXNAMELENGTH],
                char     OutFileName[MAXNAMELENGTH],
                char     ClosestApproachFileName[MAXNAMELENGTH],
                int      &InFileLength,
                int      &OutFileLength,
                int      &ClosestApproachFileLength,
                struct   Aircraft &ABLPlatform,
                int      ReferenceHour,
                int      ReferenceMinute,
                double   ReferenceSecond,
                double   RefModJulianDate,
                int      CalcYear,
                int      CalcMonth,
                int      CalcDay,
                int      CalcHour,
                int      CalcMinute,
                double   CalcSecond,
                double   LazeDuration,
                double   LaserAzimuthInDegrees,
                double   LaserAzimuthDot,
                double   LaserAzimuthDotDot,
                double   LaserElevationInDegrees,
                double   LaserElevationDot,
                double   LaserElevationDotDot,
                double   SatPositionErrorInMeters,
                double   PlatformPositionErrorInMeters,
                double   MissilePositionErrorInMeters,
                double   RangeToMissileInKilometers,
                double   OtherErrorAngleInDeg,
                double   SecondsFromVertex,
                double   InterpolationIncrement,
                double   &ThetaGInDegrees,
                ErrorStructure   &ErrorList)
```

```
{

/*******************************/
/*  VARIABLE DECLARATIONS      */
/*******************************/
    SatStructure    *SatArray = new SatStructure;
    Satellite* Sat;
        Sat = new Satellite;

    FILE             *TLEOutFile;
    FILE             *ClosestApproachFile;
    double   ThetaGInRadians;
    double   *ThetaPtr = &ThetaGInRadians;
    int      i;
    double   JulianDate;
    double   *JulianDatePtr = &JulianDate;
    double RangeToSatInKilometers;
    double *RangeToSatInKilometersPtr = &RangeToSatInKilometers;
    double ErrorAngleInRadians;
    double *ErrorAngleInRadiansPtr = &ErrorAngleInRadians;
    double SeparationAngle;
    double *SeparationAnglePtr = &SeparationAngle;
    double SepAngleDot;
    double *SepAngleDotPtr = &SepAngleDot;
    double SepAngleDotDot;
    double *SepAngleDotDotPtr = &SepAngleDotDot;
    int     Intersection;
    int     *IntersectionPtr = &Intersection;
    int     Interpolation;
    int     *InterpolationPtr = &Interpolation;
    double ClosestApproachInDegrees;
    double *ClosestApproachInDegreesPtr = &ClosestApproachInDegrees;
    double TimeToIntersect;
    double *TimeToIntersectPtr = &TimeToIntersect;


/****************************************************/
/*  INITIALIZE OUTPUT VARIABLES                     */
/****************************************************/
    InFileLength = 0;
    OutFileLength = 0;
    ThetaGInDegrees = 0.0;


/****************************************************/
/*  READ ALL SATELLITES FROM THE FILE               */
/****************************************************/
    ReadTLEFile(InFileName,
                *SatArray,
                ErrorList);

/****************************************************/
/*  DETERMINE THE NUMBER OF SATELLITES IN THE FILE */
/****************************************************/
    InFileLength = SatArray->NumSats;

/****************************************************/
/*  OPEN BOTH OUPUT FILES.  OutFileName FILE WILL  */
/*  HOLD ALL SATELLITES THAT ARE ACTUALLY          */
/*  DETERMINED TO BE INTERSECTED BY THE LASER.     */
/*  ClosestApproachFileName WILL HOLD ANY SATS     */
/*  THAT COME CLOSE ENOUGH TO THE LASER PATH TO BE */
/*  INTERPOLATED.                                  */
```

```
/****************************************************/
    if ((TLEOutFile = fopen(OutFileName, "w"))==NULL)
    {    ErrorList.AddError("PAProcessor",
                             "Cannot open TLE Output File",
                              1);
    }
    if ((ClosestApproachFile = fopen(ClosestApproachFileName, "w"))==NULL)
    {    ErrorList.AddError("PAProcessor",
                             "Cannot open TLE Output File",
                              1);
    }

/************************************************/
/*   BEGIN CALCULATIONS UNLESS CRITICAL ERROR    */
/************************************************/
    if (ErrorList.CriticalError())
        return 0;

/****************************************************/
/*  FIND THE CURRENT ANGLE OF THETA G AT THE        */
/*  TIME OF PROPAGATION                             */
/****************************************************/
    ThetaGInRadians = 0;
    FindThetaG( ReferenceHour,
                ReferenceMinute,
                ReferenceSecond,
                RefModJulianDate,
                CalcYear,
                CalcMonth,
                CalcDay,
                CalcHour,
                CalcMinute,
                CalcSecond,
                *ThetaPtr,
                ErrorList);

    ThetaGInDegrees = ThetaGInRadians * RADTODEGREES;

/************************************************/
/*   CONTINUE UNLESS CRITICAL ERROR              */
/************************************************/
    if (ErrorList.CriticalError())
        return 0;


/****************************************************/
/*  CONVERT THE PROPAGATION TIME TO A JULIAN DATE   */
/*  THAT CAN BE RECOGNIZED BY "ProcessSatellite".  */
/****************************************************/
    ConvertCalenderToJulian(CalcYear,
                            CalcMonth,
                            CalcDay,
                            CalcHour,
                            CalcMinute,
                            CalcSecond,
                            *JulianDatePtr,
                            ErrorList);

/****************************************************/
/*  PROCESS EACH SATELLITE IN ORDER AND DETERMINE    */
/*  IF IT IS INTERSECTED BY THE PLATFORM.  IF IT IS,*/
/*  THEN ADD IT TO THE OUTFILE, IF NOT, DISCARD THE */
/*  EPHEMERIS AND MOVE ON.                          */
```

```
/*****************************************************/
    OutFileLength = 0;
    for (i=0; i<SatArray->NumSats; i++)
    {
        *Sat = SatArray->Sat[i];
        Intersection = 0;

/***************************************************/
/*  CALL "ProcessSatellite" MODULE TO FIND THE     */
/*  INTERSECTION ANGLES AND TIME                   */
/***************************************************/
    ProcessSatellite(ABLPlatform,
                     *Sat,
                     ReferenceHour,
                     ReferenceMinute,
                     ReferenceSecond,
                     RefModJulianDate,
                     SecondsFromVertex,
                     InterpolationIncrement,
                     *ThetaPtr,
                     JulianDate,
                     LazeDuration,
                     LaserAzimuthInDegrees,
                     LaserAzimuthDot,
                     LaserAzimuthDotDot,
                     LaserElevationInDegrees,
                     LaserElevationDot,
                     LaserElevationDotDot,
                     SatPositionErrorInMeters,
                     PlatformPositionErrorInMeters,
                     MissilePositionErrorInMeters,
                     RangeToMissileInKilometers,
                     OtherErrorAngleInDeg,
                     *RangeToSatInKilometersPtr,
                     *ErrorAngleInRadiansPtr,
                     *SeparationAnglePtr,
                     *SepAngleDotPtr,
                     *SepAngleDotDotPtr,
                     *IntersectionPtr,
                     *InterpolationPtr,
                     *TimeToIntersectPtr,
                     *ClosestApproachInDegreesPtr,
                     ErrorList);

/************************************************/
/*  IF AN INTERSECTION OCCURS, PUT IT IN THE    */
/*  INTERSECTION OUTPUT FILE.                   */
/************************************************/
        if (Intersection == 1)
        {   OutFileLength = OutFileLength + 1;
            fputs(Sat->GetTLELine1(), TLEOutFile);
            fputs(Sat->GetTLELine2(), TLEOutFile);
        }
/************************************************/
/*  IF AN INTERPOLATION OCCURS, PUT IT IN THE   */
/*  CLOSE APPROACH OUTPUT FILE.                 */
/************************************************/
        if (Interpolation == 1)
        {   ClosestApproachFileLength = ClosestApproachFileLength + 1;
            fputs(Sat->GetTLELine1(), ClosestApproachFile);
            fputs(Sat->GetTLELine2(), ClosestApproachFile);
        }
```

```
/**************************************************/
/*   CONTINUE UNLESS CRITICAL ERROR               */
/**************************************************/
        if (ErrorList.CriticalError())
            return 0;

    }
/**********************/
/*  CLOSE THE FILES   */
/**********************/
    fclose(TLEOutFile);
    fclose(ClosestApproachFile);

    return 0;
}
```

D.6 PAPreprocessor.cpp

```
/***************************************************************************/
/*  MODULE NAME:     PAPreprocessor.cpp
*/
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    August 18, 1998                                        */
/*                                                                          */
/*  PURPOSE:         This set of modules composes the preprocessor          */
/*                   used to evaluate whether or not the satellites are ever */
/*                   above the platform horizon.                            */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/***************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "TimeModules.h"
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisModules.h"
#include "PAPreprocessor.h"
/**********************************/
/* C STANDARD LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/***************************************************************************/
/***********************       FUCTIONS        *****************************/
/***************************************************************************/


/***************************************************************************/
/*  FUNCTION NAME:   PAPreprocessor                                         */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    October 6, 1998                                        */
/*                                                                          */
/*  PURPOSE:         This procedure will read in an input file of Two Line  */
/*                   Element (TLE) sets and perform an analysis to determine */
/*                   whether or not they are within view of the airborne    */
/*                   platform.  If a satellite is in view, it will be added */
/*                   to the ouput file, which is the input file for the main */
/*                   processor.                                             */
/*                                                                          */
/*  INPUTS:          NAME:                 DEFINITION:                      */
/*                   InFileName            Holds name of the satellite file*/
/*                   OutFileName           File that holds the sats in view*/
```

```
/*                  InFileLength            The total number
/*                  ABLPlatform             Holds all information about ABL */
/*                                          Platform position/disposition   */
/*                  ReferenceHour           This holds the value of Theta G */
/*                                          at RefModJulianDate.  The angle */
/*                                          of Theta G is given in hours,   */
/*                                          minutes, and seconds instead of */
/*                                          degrees, where 24 hrs = 360 deg */
/*                  ReferenceMinute         Holds the minutes of Theta G at */
/*                                          RefModJulianDate.               */
/*                  ReferenceSecond         Holds the seconds of Theta G at */
/*                                          RefModJulianDate.               */
/*                  RefModJulianDate        This is the reference date when */
/*                                          an actual observation of the    */
/*                                          true value of theta G was made. */
/*                  CalcYear                Holds the current calender year */
/*                  Calcmonth               Holds the Calender month(1 - 12)*/
/*                  CalcDay                 Holds calender day              */
/*                  CalcHour                Holds the calender hour         */
/*                  CalcMinute              Holds the calender minute       */
/*                  CalcSecond              Holds the calender second       */
/*                  TimeToNextRun           The amount of time for which the*/
/*                                          current run must last.  This is */
/*                                          To determine how much time in   */
/*                                          seconds will transpire before   */
/*                                          next update is received.        */
/*                                                                          */
/*  OUTPUTS:        NAME:                   DESCRIPTION:                    */
/*                  InFileLength            The total number of satellites  */
/*                                          that have been evaluated in the */
/*                                          InFile                          */
/*                  OutFileLength           The total number of satellites  */
/*                                          that are in view of the platform*/
/*                                          and have been put in the outfile*/
/*                  ThetaGInDegrees         The rotation angle between the  */
/*                                          Earth's current ECEF position   */
/*                                          and its ECI position.           */
/*                  ErrorList               Errors that have occured        */
/*                                                                          */
/*                  THE FINAL OUTPUT IS THE ACTUAL OUTFILE ITSELF WHICH IS  */
/*                  WRITTEN DIRECTLY TO DISK SO IT CAN BE ACCESSED BY THE   */
/*                  MAIN PROCESSOR.                                         */
/*                                                                          */
/*  COMPILER:       Borland C++ Builder3 Standard version                   */
/*                  This compiler should be used to compile and link.       */
/*                                                                          */
/****************************************************************************/
PAPreprocessor( char    InFileName[MAXNAMELENGTH],
                char    OutFileName[MAXNAMELENGTH],
                int     &InFileLength,
                int     &OutFileLength,
                struct  Aircraft &ABLPlatform,
                int     ReferenceHour,
                int     ReferenceMinute,
                double  ReferenceSecond,
                double  RefModJulianDate,
                int     CalcYear,
                int     CalcMonth,
                int     CalcDay,
                int     CalcHour,
                int     CalcMinute,
                double  CalcSecond,
                double  TimeToNextRun,
```

```
                double  &ThetaGInDegrees,
                ErrorStructure   &ErrorList)

{
/*******************************/
/*  VARIABLE DECLARATIONS      */
/*******************************/
    SatStructure    *SatArray = new SatStructure;
    Satellite* Sat;
        Sat = new Satellite;
    FILE            *TLEOutFile;
    int     SatelliteInView;
    int     *SatInViewPtr = &SatelliteInView;
    int     OrbitInView;
    int     *OrbitInViewPtr = &OrbitInView;
    double  ThetaGInRadians;
    double  *ThetaPtr = &ThetaGInRadians;
    int     i;
    double  JulianDate;
    double  *JulianDatePtr = &JulianDate;
    double  Inclination;
    double  *InclinationPtr = &Inclination;
    double  RightAscension;
    double  *RightAscensionPtr = &RightAscension;
    double  Eccentricity;
    double  *EccentricityPtr = &Eccentricity;
    double  MeanMotion;
    double  *MeanMotionPtr = &MeanMotion;
    double  ArgumentOfPerigee;
    double  *ArgumentOfPerigeePtr = &ArgumentOfPerigee;
    double  MeanAnomaly;
    double  *MeanAnomalyPtr = &MeanAnomaly;
    double SatX;
    double *SatXPtr = &SatX;
    double SatY;
    double *SatYPtr = &SatY;
    double SatZ;
    double *SatZPtr = &SatZ;
    double SatXdot;
    double *SatXdotPtr = &SatXdot;
    double SatYdot;
    double *SatYdotPtr = &SatYdot;
    double SatZdot;
    double *SatZdotPtr = &SatZdot;
    double Delta;
    double *DeltaPtr = &Delta;
    double TimeToRise;
    double *TimeToRisePtr = &TimeToRise;
    double Dvector;
    double *DvectorPtr = &Dvector;
    double  CriticalRadius;
    double  *CriticalRadiusPtr = &CriticalRadius;
    double  SatRadius;
    double  *SatRadiusPtr = &SatRadius;


/**************************************************/
/*  INITIALIZE OUTPUT VARIABLES                   */
/**************************************************/
    InFileLength = 0;
    OutFileLength = 0;
    ThetaGInDegrees = 0.0;
```

```
/***************************************************/
/*  READ ALL SATELLITES FROM THE FILE              */
/***************************************************/
    ReadTLEFile(InFileName,
                *SatArray,
                ErrorList);

/***************************************************/
/*  DETERMINE THE NUMBER OF SATELLITES IN THE FILE */
/***************************************************/
    InFileLength = SatArray->NumSats;

    if ((TLEOutFile = fopen(OutFileName, "w"))==NULL)
    {     ErrorList.AddError("PAProcessor",
                             "Cannot open TLE Output File",
                              1);
    }

/************************************************/
/*   BEGIN CALCULATIONS UNLESS CRITICAL ERROR   */
/************************************************/
    if (ErrorList.CriticalError())
        return 0;

/***************************************************/
/*  FIND THE CURRENT ANGLE OF THETA G AT THE       */
/*  TIME OF PROPAGATION                            */
/***************************************************/
    ThetaGInRadians = 0;
    FindThetaG( ReferenceHour,
                ReferenceMinute,
                ReferenceSecond,
                RefModJulianDate,
                CalcYear,
                CalcMonth,
                CalcDay,
                CalcHour,
                CalcMinute,
                CalcSecond,
                *ThetaPtr,
                ErrorList);


/************************************************/
/*   CONTINUE UNLESS CRITICAL ERROR             */
/************************************************/
    if (ErrorList.CriticalError())
        return 0;


/***************************************************/
/*  CONVERT THE PROPAGATION TIME TO A JULIAN DATE  */
/*  THAT CAN BE RECOGNIZED BY "EvaluateEphemeris". */
/***************************************************/
    ConvertCalenderToJulian(CalcYear,
                            CalcMonth,
                            CalcDay,
                            CalcHour,
                            CalcMinute,
                            CalcSecond,
                            *JulianDatePtr,
                            ErrorList);
```

```
/***************************************************/
/*  PROCESS EACH SATELLITE IN ORDER AND DETERMINE   */
/*  IF IT IS IN VIEW OF THE PLATFORM.  IF IT IS,    */
/*  THEN ADD IT TO THE OUTFILE, IF NOT, DISCARD THE */
/*  EPHEMERIS AND MOVE ON.                          */
/***************************************************/
    OutFileLength = 0;
    for (i=0; i<SatArray->NumSats; i++)
    {
        *Sat = SatArray->Sat[i];
        SatelliteInView = 0;

        EvaluateEphemeris(  *Sat,
                            ABLPlatform,
                            ThetaGInRadians,
                            JulianDate,
                            TimeToNextRun,
                            *SatInViewPtr,
                            *OrbitInViewPtr,
                            *SatXPtr,
                            *SatYPtr,
                            *SatZPtr,
                            *SatXdotPtr,
                            *SatYdotPtr,
                            *SatZdotPtr,
                            *DeltaPtr,
                            *InclinationPtr,
                            *RightAscensionPtr,
                            *EccentricityPtr,
                            *MeanMotionPtr,
                            *ArgumentOfPerigeePtr,
                            *MeanAnomalyPtr,
                            *DvectorPtr,
                            *TimeToRisePtr,
                            *CriticalRadiusPtr,
                            *SatRadiusPtr,
                            ErrorList);


        if (SatelliteInView == 1)
        {   OutFileLength = OutFileLength + 1;
            fputs(Sat->GetTLELine1(), TLEOutFile);
            fputs(Sat->GetTLELine2(), TLEOutFile);
        }

        ThetaGInDegrees = ThetaGInRadians * RADTODEGREES;

/***********************************************/
/*   CONTINUE UNLESS CRITICAL ERROR            */
/***********************************************/
        if (ErrorList.CriticalError())
            return 0;

    }
    fclose(TLEOutFile);

    return 0;
}
```

D.7 ProcessSatellite.cpp

```
/****************************************************************************/
/*  MODULE NAME:     ProcessSatellite.cpp                                   */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    14 January, 1999                                       */
/*                                                                          */
/*  PURPOSE:         This module supports the meat of the Main Processor and */
/*                   is used to evaluate the error angle and the displacement*/
/*                   angle between the laser position vector in the REN frame*/
/*                   and the satellite position vector in the same frame.  It*/
/*                   uses this angle and its rate of change to determine when*/
/*                   and if the satellite will intersect the path of the    */
/*                   laser.                                                 */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/****************************************************************************/
/*********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/*********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/*********************************/
/* USER-BUILT LIBRARIES          */
/*********************************/
#include "TimeModules.h"
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisModules.h"
#include "SGP4SupportModules.h"
#include "FindDisplacementAngleModules.h"
#include "TargetSatellite.h"
#include "TargetPlatform.h"
#include "TargetLaser.h"
#include "ProcessSatellite.h"
/*********************************/
/* C STANDARD LIBRARIES          */
/*********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/****************************************************************************/
/***********************          FUCTIONS         *************************/
/****************************************************************************/

/****************************************************************************/
/*  FUNCTION NAME:  ProcessSatellite                                        */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   January 13, 1999                                        */
/*                                                                          */
/*  PURPOSE:        This module supports the meat of theMain Processor and  */
/*                  is used to evaluate the error angle and the displacement*/
```

```
/*                  angle between the laser position vector in the REN frame*/
/*                  and the satellite position vector in the same frame.  It*/
/*                  uses this angle and its rate of change to determine when*/
/*                  and if the satellite will intersect the path of the     */
/*                  laser.                                                  */
/*                                                                          */
/*  INPUTS:         NAME:                   DEFINITION:                     */
/*                  Sat                     Holds all ephemeris information */
/*                                          for the Satellite being studied */
/*                  ABLPlatform             Holds all information about ABL */
/*                                          Platform position/disposition   */
/*                  JulianDate              The time to which the position  */
/*                                          of sat should be propagated to  */
/*                  ThetaGInRadians         The angle between the Greenwich */
/*                                          Meridian and the Vernal Equinox */
/*                                          at JulianDate.                  */
/*                  LazeDuration            The amount of time in seconds   */
/*                                          that the laser will be on.      */
/*                  LazerAzimuthInDegrees   Lazer Azimuth at Laze Start time*/
/*                                          in Degrees                      */
/*                  LazerAzimuthDot         The rate of change of the Az    */
/*                                          in Degrees/Sec.                 */
/*                  LazerAzimuthDotDot      The rate of change of the rate  */
/*                                          of change of the Azimuth (Accel)*/
/*                                          in Degrees/Sec^2                */
/*                  LazerElevationInDegrees Lazer Elevation at Laze Start   */
/*                                          in Degrees                      */
/*                  LazerElevationDot       The rate of change of the El    */
/*                                          in Degrees/Sec.                 */
/*                  LazerElevationDotDot    The rate of change of the rate  */
/*                                          of change of the Elevat. (Accel)*/
/*                                          in Degrees/Sec^2                */
/*                  SatPositionErrorInMeters Holds the radius of the error  */
/*                                          spheroid that describes the     */
/*                                          area in which the satellite is  */
/*                                          known to exist (in meters).     */
/*                  PlatformPositionError...Holds the radius of the error   */
/*                                          spheroid that describes the     */
/*                                          area in which the platform is   */
/*                                          known to exist (in meters).     */
/*                  MissilePositionError... Holds the radius of the error   */
/*                                          spheroid that describes the     */
/*                                          area in which the missile is    */
/*                                          known to exist (in meters).     */
/*                  RangeToMissileInKilo... The Range to the missile (km)   */
/*                  OtherErrorAnglesInDeg   Holds any other error angles    */
/*                                          (in degrees) that may be a      */
/*                                          significant source of error.    */
/*                                          This should usually be set to   */
/*                                          zero (0.0) float.               */
/*  OUTPUTS:        NAME:                   DESCRIPTION:                    */
/*                  RangeInKilometers       Holds the range of the aircraft */
/*                                          to the satellite in kilometers. */
/*                  ErrorAngleInRadians     The total error angle in radians*/
/*                  SeparationAngle         The separation (in radians) of  */
/*                                          the LaserRENRho and             */
/*                                          PlatformSatRENRho vectors.      */
/*                  SeparationAngleDot      The rate of change (in rad/sec) */
/*                                          of the separation of LaserRENRho*/
/*                                          PlatformSatRENRho vectors.      */
/*                  SeparationAngleDotDot   The acceleration (in rad/sec^2) */
/*                                          of the separation of LaserRENRho*/
/*                                          and PlatformSatRENRho vectors.  */
```

```
/*                  Intersection            Will the laser intersect this   */
/*                                          satellite? 1=YES, 2=NO          */
/*                  TimeToIntersect         How much time (in seconds) is   */
/*                                          forecasted to go by before the  */
/*                                          laser intersects the satellite. */
/*                  ErrorList               The Errors which have occurred  */
/*                                                                          */
/*  COMPILER:       Borland C++ Builder3 Standard version                   */
/*                  This compiler should be used to compile and link.       */
/*                                                                          */
/****************************************************************************/
void ProcessSatellite(struct Aircraft &Platform,
                      struct Satellite &Sat,
                      int    ReferenceHour,
                      int    ReferenceMinute,
                      double ReferenceSecond,
                      double RefModJulianDate,
                      double SecondsFromVertex,
                      double InterpolationIncrement,
                      double &ThetaGInRad,
                      double JulianDate,
                      double LazeDuration,
                      double LaserAzimuthInDegrees,
                      double LaserAzimuthDot,
                      double LaserAzimuthDotDot,
                      double LaserElevationInDegrees,
                      double LaserElevationDot,
                      double LaserElevationDotDot,
                      double SatPositionErrorInMeters,
                      double PlatformPositionErrorInMeters,
                      double MissilePositionErrorInMeters,
                      double RangeToMissileInKilometers,
                      double OtherErrorAngleInDeg,
                      double &RangeInKilometers,
                      double &ErrorAngleInRadians,
                      double &SeparationAngle,
                      double &SepAngleDot,
                      double &SepAngleDotDot,
                      int    &Intersection,
                      int    &Interpolation,
                      double &TimeToIntersect,
                      double &ClosestApproachInDegrees,
                      ErrorStructure   &ErrorList)
{
/*******************************/
/*  VARIABLE DECLARATIONS      */
/*******************************/
    double Check;
    double QuadraticSolnOne;
    double QuadraticSolnTwo;
    double PlatformSatRENRhoR;
    double *PlatformSatRENRhoRPtr = &PlatformSatRENRhoR;
    double PlatformSatRENRhoE;
    double *PlatformSatRENRhoEPtr = &PlatformSatRENRhoE;
    double PlatformSatRENRhoN;
    double *PlatformSatRENRhoNPtr = &PlatformSatRENRhoN;
    double PlatformSatRENRhoRDot;
    double *PlatformSatRENRhoRDotPtr = &PlatformSatRENRhoRDot;
    double PlatformSatRENRhoEDot;
    double *PlatformSatRENRhoEDotPtr = &PlatformSatRENRhoEDot;
    double PlatformSatRENRhoNDot;
    double *PlatformSatRENRhoNDotPtr = &PlatformSatRENRhoNDot;
    double PlatformSatRENRhoRDotDot;
```

```
    double *PlatformSatRENRhoRDotDotPtr = &PlatformSatRENRhoRDotDot;
    double PlatformSatRENRhoEDotDot;
    double *PlatformSatRENRhoEDotDotPtr = &PlatformSatRENRhoEDotDot;
    double PlatformSatRENRhoNDotDot;
    double *PlatformSatRENRhoNDotDotPtr = &PlatformSatRENRhoNDotDot;
    double LaserRENRhoR;
    double *LaserRENRhoRPtr = &LaserRENRhoR;
    double LaserRENRhoE;
    double *LaserRENRhoEPtr = &LaserRENRhoE;
    double LaserRENRhoN;
    double *LaserRENRhoNPtr = &LaserRENRhoN;
    double LaserRENRhoRDot;
    double *LaserRENRhoRDotPtr = &LaserRENRhoRDot;
    double LaserRENRhoEDot;
    double *LaserRENRhoEDotPtr = &LaserRENRhoEDot;
    double LaserRENRhoNDot;
    double *LaserRENRhoNDotPtr = &LaserRENRhoNDot;
    double LaserRENRhoRDotDot;
    double *LaserRENRhoRDotDotPtr = &LaserRENRhoRDotDot;
    double LaserRENRhoEDotDot;
    double *LaserRENRhoEDotDotPtr = &LaserRENRhoEDotDot;
    double LaserRENRhoNDotDot;
    double *LaserRENRhoNDotDotPtr = &LaserRENRhoNDotDot;
    char   buffer[MAXMESSAGELENGTH] = " ";

/*****************************************************/
/*  FIND THE SEPARATION ANGLE BETWEEN THE SATELLITE  */
/*  AND LASER POSITION VECTORS.  ALSO, FIND THE RATE */
/*  OF CHANGE AND ACCELERATION OF THIS ANGLE AS      */
/*  NEARLY AS POSSIBLE GIVEN THE PREEXISTING         */
/*  CONDITIONS.                                      */
/*****************************************************/
FindDisplacementAngles(Platform,
                       Sat,
                       ThetaGInRad,
                       JulianDate,
                       LaserAzimuthInDegrees,
                       LaserAzimuthDot,
                       LaserAzimuthDotDot,
                       LaserElevationInDegrees,
                       LaserElevationDot,
                       LaserElevationDotDot,
                       SatPositionErrorInMeters,
                       PlatformPositionErrorInMeters,
                       MissilePositionErrorInMeters,
                       RangeToMissileInKilometers,
                       OtherErrorAngleInDeg,
                       *PlatformSatRENRhoRPtr,
                       *PlatformSatRENRhoEPtr,
                       *PlatformSatRENRhoNPtr,
                       *PlatformSatRENRhoRDotPtr,
                       *PlatformSatRENRhoEDotPtr,
                       *PlatformSatRENRhoNDotPtr,
                       *PlatformSatRENRhoRDotDotPtr,
                       *PlatformSatRENRhoEDotDotPtr,
                       *PlatformSatRENRhoNDotDotPtr,
                       *LaserRENRhoRPtr,
                       *LaserRENRhoEPtr,
                       *LaserRENRhoNPtr,
                       *LaserRENRhoRDotPtr,
                       *LaserRENRhoEDotPtr,
                       *LaserRENRhoNDotPtr,
                       *LaserRENRhoRDotDotPtr,
```

```
                          *LaserRENRhoEDotDotPtr,
                          *LaserRENRhoNDotDotPtr,
                          RangeInKilometers,
                          ErrorAngleInRadians,
                          SeparationAngle,
                          SepAngleDot,
                          SepAngleDotDot,
                          ErrorList);


/*************************************************/
/*   IF ACCELERATION IS ZERO, THEN AN ERROR HAS */
/*   ALMOST CERTAINLY OCCURRED.  TRAP THIS ERROR*/
/*   AND NOTIFY THE USER.                       */
/*************************************************/
    if (SepAngleDotDot == 0.0)
    {   sprintf(buffer,"Satellite SSC: %d,Accel. is zero...Unable to
calculate",
                        Sat.GetSSCNumber());
        ErrorList.AddError("ProcessSatellite",
                            buffer,
                            1);
        return;
    }

/*************************************************/
/*  IF THE SEPARATION ANGLE IS CURRENTLY        */
/*  SMALLER THAT THE ERROR ANGLE, THEN THE SAT  */
/*  IS CURRENTLY BEING INTERSECTED BY THE BEAM. */
/*************************************************/
    if (SeparationAngle <=ErrorAngleInRadians)
    {   Intersection = 1;
        TimeToIntersect = 0.0;
    }

/*************************************************/
/*  OTHERWISE, USE THE QUADRATIC FORMULA TO FIND*/
/*  THE ROOTS TO THE TAYLOR SERIES EXPANSION OF */
/*  THE SEPARTION ANGLE. (IE:                   */
/*                                              */
/*  B = SEP ANGLE                               */
/*  BDot = RATE OF CHANGE OF SEP ANGLE          */
/*  BDotDot = ACCEL OF SEPARATION ANGLE         */
/*  A = ERROR ANGLE DESCRIBING POSITION ERROR OF*/
/*      SATELLITE.                              */
/*  T = TIME ELAPSED                            */
/*                                              */
/*  TAYLOR'S SERIES TO SECOND DEGREE            */
/*                                              */
/*  E = B + BDot*T + (1/2)BDotDot(T^2)          */
/*  (or)                                        */
/*  0 = (B-E) + BDot*T + (1/2)*BDotDot*(T^2)    */
/*                                              */
/*  TO FIND TIMES THAT SATISFY INTERSECTION     */
/*  OF LASER WITH SAT, APPLY QUADRATIC EQUATION */
/*  TO TAYLOR'S EXPANSION WITH:                 */
/*                                              */
/*  A = (1/2)*BDotDot                           */
/*  B = BDot                                    */
/*  C = (B-E)                                   */
/*************************************************/

    else
```

```
/*************************************************/
/*  FIRST FIND:                                  */
/*                                               */
/*  sqrt(B^2 -4AC)                               */
/*************************************************/
    {   Check = pow(SepAngleDot,2) -
                2.0 * SepAngleDotDot *
                (SeparationAngle - ErrorAngleInRadians);

/*************************************************/
/*   IF INSIDE sqrt IS NEGATIVE, THEN NO REAL    */
/*   ROOTS, AND THERE WILL BE NO INTERSECTION    */
/*************************************************/
        if (Check < 0.0)
        {   Intersection = 0;
            TimeToIntersect = 0.0;
        }

/*************************************************/
/*  OTHERWISE, FIND BOTH QUADRATIC ROOTS, AND    */
/*  USE THE ONE THAT IS CLOSEST IN THE FUTURE    */
/*  (IE: THE ONE THAT IS LEAST POSITIVE, BUT NOT*/
/*  NEGATIVE.                                    */
/*************************************************/
        else
        {   QuadraticSolnOne = (-SepAngleDot + sqrt(Check)) /
                                SepAngleDotDot;
            QuadraticSolnTwo = (-SepAngleDot - sqrt(Check)) /
                                SepAngleDotDot;
            if ((QuadraticSolnOne > 0.0)&&(QuadraticSolnTwo > 0.0))
            {   if (QuadraticSolnOne > QuadraticSolnTwo)
                    TimeToIntersect = QuadraticSolnTwo;
                else
                    TimeToIntersect = QuadraticSolnOne;
            }
            else if (QuadraticSolnOne > 0.0)
                    TimeToIntersect = QuadraticSolnOne;
            else if (QuadraticSolnTwo > 0.0)
                    TimeToIntersect = QuadraticSolnTwo;
            else TimeToIntersect = QuadraticSolnTwo;

/*************************************************/
/*  NOW, COMPARE THIS FUTURE TIME WITH THE       */
/*  DURATION OF THE LAZE TIME, IF DURATION IS    */
/*  LARGER THAN INTERSECTION TIME, THEN AN       */
/*  INTERSECTION SHOULD THEORETICALLY OCCUR.     */
/*                                               */
/*  NOTE!::                                      */
/*  USE CAUTION WITH THIS FORECASTING TECHNIQUE.*/
/*  THIS ASSUMES THAT THE INITIAL CONDITIONS     */
/*  TRUE THROUGHOUT THE LAZE, WHICH PROBABLY     */
/*  WILL NOT HAPPEN.  THEREFORE, THE FORECAST    */
/*  MAY DEVIATE FROM REALITY MORE AND MORE AS    */
/*  LAZE DURATION AND ACCELERATIONS ARE          */
/*  INCREASED.                                   */
/*************************************************/
            if (TimeToIntersect < 0.0)
                Intersection = 0;
            else
            {   if (TimeToIntersect > LazeDuration)
                    Intersection = 0;
                else
                    Intersection = 1;
```

```
            }
        }
    }

 /***************************************************/
 /*  JUST BECAUSE AN INTERSECTION WAS "FORECASTED"   */
 /*  USING INITIAL CONDITIONS, DOES NOT MEAN AN      */
 /*  INTERSECTION WILL OCCUR.  THE ABOVE FORECAST    */
 /*  IS ONLY A ROUGH APPROXIMATION.  NOW, IF A       */
 /*  INTERSECTION IS FORECASTED, WE WILL ACTUALLY    */
 /*  STUDY THE LOCATION OF THE LASER BEAM AND THE    */
 /*  FOR A GIVEN TIME "SecondsBeforeVertex" AND      */
 /*  ACTUALLY STEP THROUGH BY TIME INCREMENTS OF     */
 /*  LENGTH "InterpolationIncrement" TO SEE IF THE   */
 /*  LASER ACTUALLY GETS CLOSE ENOUGH TO INTERSECT.  */
 /*  THIS CANNOT BE DONE FOR EVERY SATELLITE,        */
 /*  BECAUSE THE CALCULATIONS ARE TIME CONSUMING.    */
 /*  INTERPOLATION IS DONE BY SLIGHTLY MODIFYING     */
 /*  THE TARGETING MODULES TO ACCEPT SLIGHT POSITION*/
 /*  CHANGES TO REFLECT TIME PASSING.  THIS ALL      */
 /*  ASSUMES THAT THE PLATFORM DOES NOT CHANGE       */
 /*  COURSE OR ACCELERATE MID-FIRE.                  */
 /***************************************************/
    if (Intersection)
    {   Interpolation = 1;
        InterpolateVertex(Platform,
                          Sat,
                          ReferenceHour,
                          ReferenceMinute,
                          ReferenceSecond,
                          RefModJulianDate,
                          JulianDate,
                          LazeDuration,
                          LaserAzimuthInDegrees,
                          LaserAzimuthDot,
                          LaserAzimuthDotDot,
                          LaserElevationInDegrees,
                          LaserElevationDot,
                          LaserElevationDotDot,
                          ErrorAngleInRadians,
                          SecondsFromVertex,
                          InterpolationIncrement,
                          TimeToIntersect,
                          ClosestApproachInDegrees,
                          ErrorList);

        if ((ClosestApproachInDegrees*DEGTORADIANS) < ErrorAngleInRadians)
            Intersection = 1;
        else
            Intersection = 0;
            TimeToIntersect = 0.0;
    }
    else
    {   Interpolation = 0;
        TimeToIntersect = 0.0;
        ClosestApproachInDegrees = 0.0;
    }
    return;
}

/****************************************************************************/
/*  FUNCTION NAME:  InterpolateVertex                                        */
/*  AUTHOR:         Captain David Vloedman                                   */
```

```
/*  DATE CREATED:    January 13, 1999                                      */
/*                                                                         */
/*  PURPOSE:         This module supports the meat of the Main Processor and */
/*                   is used to evaluate the error angle and the displacement*/
/*                   angle between the laser position vector in the REN frame*/
/*                   and the satellite position vector in the same frame     */
/*                   during the relatively short time of estimated closest   */
/*                   approach of the two vectors.  The smaller the inter-    */
/*                   polation increment, the more accurate the estimate, and */
/*                   the longer the processing time.                         */
/*                                                                         */
/*  INPUTS:          NAME:                  DEFINITION:                    */
/*                   Sat                    Holds all ephemeris information */
/*                                          for the Satellite being studied */
/*                   ABLPlatform            Holds all information about ABL */
/*                                          Platform position/disposition   */
/*                   ReferenceHour          This holds the value of Theta G */
/*                                          at RefModJulianDate.  The angle */
/*                                          of Theta G is given in hours,   */
/*                                          minutes, and seconds instead of */
/*                                          degrees, where 24 hrs = 360 deg */
/*                   ReferenceMinute        Holds the minutes of Theta G at */
/*                                          RefModJulianDate.               */
/*                   ReferenceSecond        Holds the seconds of Theta G at */
/*                                          RefModJulianDate.               */
/*                   RefModJulianDate       This is the reference date when */
/*                                          an actual observation of the    */
/*                                          true value of theta G was made. */
/*                   JulianDate             The time to which the position  */
/*                                          of sat should be propagated to  */
/*                   ThetaGInRadians        The angle between the Greenwich */
/*                                          Meridian and the Vernal Equinox */
/*                                          at JulianDate.                  */
/*                   LazeDuration           The amount of time in seconds   */
/*                                          that the laser will be on.      */
/*                   LazerAzimuthInDegrees  Lazer Azimuth at Laze Start time*/
/*                                          in Degrees                      */
/*                   LazerAzimuthDot        The rate of change of the Az    */
/*                                          in Degrees/Sec.                 */
/*                   LazerAzimuthDotDot     The rate of change of the rate  */
/*                                          of change of the Azimuth (Accel)*/
/*                                          in Degrees/Sec^2                */
/*                   LazerElevationInDegrees Lazer Elevation at Laze Start  */
/*                                          in Degrees                      */
/*                   LazerElevationDot      The rate of change of the El    */
/*                                          in Degrees/Sec.                 */
/*                   LazerElevationDotDot   The rate of change of the rate  */
/*                                          of change of the Elevat. (Accel)*/
/*                                          in Degrees/Sec^2                */
/*                   PositionError          Holds the radius of the error   */
/*                                          spheroid that describes the     */
/*                                          area in which the satellite is  */
/*                                          known to exist (in meters).     */
/*                   OtherErrorAnglesInDeg  Holds any other error angles    */
/*                                          (in degrees) that may be a      */
/*                                          significant source of error.    */
/*                                          This should usually be set to   */
/*                                          zero (0.0) float.               */
/*                   SecondsFromVertex      This holds the amount of time   */
/*                                          before the forecasted intercept */
/*                                          time (if any) of the satellite  */
/*                                          that should be studied more     */
/*                                          closely (interpolated) to see if*/
```

```
/*                                          an intersection actually occurs */
/*                InterpolationIncrement   The length of the time step in  */
/*                                          the interpolation sequence. This*/
/*                                          is the length of time between   */
/*                                          steps.                          */
/*   OUTPUTS:      NAME:                    DESCRIPTION:                    */
/*                 RangeInKilometers        Holds the range of the aircraft */
/*                                          to the satellite in kilometers. */
/*                 ErrorAngleInRadians      The total error angle in radians*/
/*                 SeparationAngle          The separation (in radians) of  */
/*                                          the LaserRENRho and             */
/*                                          PlatformSatRENRho vectors.      */
/*                 SeparationAngleDot       The rate of change (in rad/sec) */
/*                                          of the separation of LaserRENRho*/
/*                                          PlatformSatRENRho vectors.      */
/*                 SeparationAngleDotDot    The acceleration (in rad/sec^2) */
/*                                          of the separation of LaserRENRho*/
/*                                          and PlatformSatRENRho vectors.  */
/*                 Intersection             Will the laser intersect this   */
/*                                          satellite? 1=YES, 2=NO          */
/*                 TimeToIntersect          How much time (in seconds) is   */
/*                                          forecasted to go by before the  */
/*                                          laser intersects the satellite. */
/*                 ErrorList                The Errors which have occurred  */
/*                                                                          */
/*   COMPILER:     Borland C++ Builder3 Standard version                   */
/*                 This compiler should be used to compile and link.       */
/*                                                                          */
/***************************************************************************/
void InterpolateVertex(struct Aircraft &Platform,
                       struct Satellite &Sat,
                       int    ReferenceHour,
                       int    ReferenceMinute,
                       double ReferenceSecond,
                       double RefModJulianDate,
                       double JulianDate,
                       double LazeDuration,
                       double LaserAzimuthInDegrees,
                       double LaserAzimuthDot,
                       double LaserAzimuthDotDot,
                       double LaserElevationInDegrees,
                       double LaserElevationDot,
                       double LaserElevationDotDot,
                       double ErrorAngleInRadians,
                       double SecondsFromVertex,
                       double InterpolationIncrement,
                       double &TimeToIntersect,
                       double &ClosestApproachInDegrees,
                       ErrorStructure   &ErrorList)
{   double  Dummy;
    double  *DummyPtr = &Dummy;
    double  TimeOfForecastedVertex;
    double  InterpolationStartTime;
    double  StepInterval;
    int     Continue;
    double  ThetaGInRadians;
    double  *ThetaPtr = &ThetaGInRadians;
    int     CalcYear;
    int     *CalcYearPtr = &CalcYear;
    int     CalcMonth;
    int     *CalcMonthPtr = &CalcMonth;
    int     CalcDay;
    int     *CalcDayPtr = &CalcDay;
```

```
    int      CalcHour;
    int      *CalcHourPtr = &CalcHour;
    int      CalcMinute;
    int      *CalcMinutePtr = &CalcMinute;
    double   CalcSecond;
    double   *CalcSecondPtr = &CalcSecond;
    double   ChangeInX;
    double   ChangeInY;
    double   ChangeInZ;
    double   XVelocity;
    double   YVelocity;
    double   ZVelocity;
    double   ClosestApproachInRadians;
    double   CurrentLaserAzimuthInDegrees;
    double   CurrentLaserElevationInDegrees;
    double   StepTime;
    double   LastSepAngle;
    double   TimeElapsed;
    double PlatformSatRENRhoR;
    double *PlatformSatRENRhoRPtr = &PlatformSatRENRhoR;
    double PlatformSatRENRhoE;
    double *PlatformSatRENRhoEPtr = &PlatformSatRENRhoE;
    double PlatformSatRENRhoN;
    double *PlatformSatRENRhoNPtr = &PlatformSatRENRhoN;
    double PlatformSatRENRhoRDot;
    double *PlatformSatRENRhoRDotPtr = &PlatformSatRENRhoRDot;
    double PlatformSatRENRhoEDot;
    double *PlatformSatRENRhoEDotPtr = &PlatformSatRENRhoEDot;
    double PlatformSatRENRhoNDot;
    double *PlatformSatRENRhoNDotPtr = &PlatformSatRENRhoNDot;
    double PlatformSatRENRhoRDotDot;
    double *PlatformSatRENRhoRDotDotPtr = &PlatformSatRENRhoRDotDot;
    double PlatformSatRENRhoEDotDot;
    double *PlatformSatRENRhoEDotDotPtr = &PlatformSatRENRhoEDotDot;
    double PlatformSatRENRhoNDotDot;
    double *PlatformSatRENRhoNDotDotPtr = &PlatformSatRENRhoNDotDot;
    double LaserRENRhoR;
    double *LaserRENRhoRPtr = &LaserRENRhoR;
    double LaserRENRhoE;
    double *LaserRENRhoEPtr = &LaserRENRhoE;
    double LaserRENRhoN;
    double *LaserRENRhoNPtr = &LaserRENRhoN;
    double LaserRENRhoRDot;
    double *LaserRENRhoRDotPtr = &LaserRENRhoRDot;
    double LaserRENRhoEDot;
    double *LaserRENRhoEDotPtr = &LaserRENRhoEDot;
    double LaserRENRhoNDot;
    double *LaserRENRhoNDotPtr = &LaserRENRhoNDot;
    double LaserRENRhoRDotDot;
    double *LaserRENRhoRDotDotPtr = &LaserRENRhoRDotDot;
    double LaserRENRhoEDotDot;
    double *LaserRENRhoEDotDotPtr = &LaserRENRhoEDotDot;
    double LaserRENRhoNDotDot;
    double *LaserRENRhoNDotDotPtr = &LaserRENRhoNDotDot;
    double SeparationAngle;
    double *SeparationAnglePtr = &SeparationAngle;


/*******************************************/
/*  FIND THE ACTUAL JULIAN DATE START TIME */
/*  OF THE VERTEX INTERPOLATION.           */
/*******************************************/
    TimeOfForecastedVertex = JulianDate + TimeToIntersect/SECSPER24HOURS;
```

```
    InterpolationStartTime = TimeOfForecastedVertex - SecondsFromVertex/
                                                      SECSPER24HOURS;
/**********************************************/
/*  DETERMINE THE VELOCITY (ASSUMED CONSTANT)*/
/*  OF THE AIRCRAFT.                         */
/**********************************************/
    XVelocity = Platform.GetVelocityX();
    YVelocity = Platform.GetVelocityY();
    ZVelocity = Platform.GetVelocityZ();

/*********************************************/
/*  SET THE INITIAL CONDITIONS FOR STARTING*/
/*  THE INTERPOLATION LOOP.  STEPTIME HOLDS*/
/*  THE CURRENT JULIANDATE FOR THE STEP    */
/*  BEING EVALUATED.  STEPINTERVAL IS THE  */
/*  AMOUNT OF TIME (IN JULIAN DAY UNITS)   */
/*  THAT TRANSPIRES BETWEEN STEPS. THE     */
/*  LASTSEPANGLE IS THE LAST SEPARATION    */
/*  ANGLE FOUND IN THE PREVIOUS STEP.  IT  */
/*  IS INITIALY SET TO TWO PI SO THAT THE  */
/*  NEXT ANGLE EVALUATED WILL BE LOWER.    */
/*  THE LOOP CONTINUES UNTIL THE VERTEX    */
/*  SWINGS "UP".  THAT IS, UNTIL THE LASER */
/*  AND SATELLITE ARE SEEN TO BE MOVING    */
/*  AWAY FROM EACH OTHER.  THIS WILL BE THE*/
/*  CASE WHEN THE LAST SEPARATION ANGLE    */
/*  EVALUATED IS LOWER THAN THE CURRENT    */
/*  SEPARATION ANGLE.                      */
/*********************************************/
    Continue = 1;
    StepInterval = InterpolationIncrement / SECSPER24HOURS;
    StepTime = InterpolationStartTime;
    LastSepAngle = TWOPI;

    while (Continue)
    {   /**************************************************/
        /*  FIRST, COMPUTE THE TIME THAT HAS ELAPSED IN */
        /*  THE CURRENT INTERPOLATION STEP              */
        /**************************************************/
        TimeElapsed = (StepTime - JulianDate) * SECSPER24HOURS;

        /**************************************************/
        /*  FIND THE EXACT CALENDAR DATE OF THIS        */
        /*  INTERPOLATION STEP TO PASS TO "FINDTHETAG"  */
        /**************************************************/
        ConvertJulianToCalender(*CalcYearPtr,
                                *CalcMonthPtr,
                                *CalcDayPtr,
                                *CalcHourPtr,
                                *CalcMinutePtr,
                                *CalcSecondPtr,
                                StepTime,
                                ErrorList);

        /*****************************************************/
        /*  FIND THE CURRENT ANGLE OF THETA G AT THE        */
        /*  CURRENT STEP TIME                               */
        /*****************************************************/
        ThetaGInRadians = 0;
        FindThetaG(ReferenceHour,
                   ReferenceMinute,
                   ReferenceSecond,
                   RefModJulianDate,
```

```
            CalcYear,
            CalcMonth,
            CalcDay,
            CalcHour,
            CalcMinute,
            CalcSecond,
            *ThetaPtr,
            ErrorList);

/************************************************/
/*  FIND CHANGE IN PLATFORM POSITION (ECEF)     */
/************************************************/
ChangeInX = TimeElapsed * XVelocity / 3600;
ChangeInY = TimeElapsed * YVelocity / 3600;
ChangeInZ = TimeElapsed * ZVelocity / 3600;

/************************************************/
/*  FIND CHANGE IN LAZER POSITION.  FIRST THE   */
/*  AZIMUTH.  NOTE THAT IF THE AZIMUTH CROSSES  */
/*  360 DEGREES, IT IS RESET TO ZERO.           */
/************************************************/
CurrentLaserAzimuthInDegrees = LaserAzimuthInDegrees +
                        TimeElapsed * LaserAzimuthDot +
                        (0.50) * LaserAzimuthDotDot *
                        pow(TimeElapsed, 2.0);
if (CurrentLaserAzimuthInDegrees > 360.0)
   CurrentLaserAzimuthInDegrees = CurrentLaserAzimuthInDegrees - 360.0;

/************************************************/
/*  NOW FIND THE CHANGE IN ELEVATION.  NOTE THAT*/
/*  IF THE ELEVATION SWINGS PAST 90 DEGREES (NOT*/
/*  LIKELY IN OPERATIONAL WORLD) THE ELEVATION  */
/*  BEGINS SWINGING BACK TOWARD 0 DEGREES, AND  */
/*  THE AZIMUTH SWINGS AROUND 180 DEGREES.      */
/************************************************/
CurrentLaserElevationInDegrees = LaserElevationInDegrees +
                        TimeElapsed * LaserElevationDot +
                        (0.50) * LaserElevationDotDot *
                        pow(TimeElapsed, 2.0);
if (CurrentLaserElevationInDegrees > 90.0)
{   CurrentLaserAzimuthInDegrees = CurrentLaserAzimuthInDegrees +
                                   180.0;
    CurrentLaserElevationInDegrees = 90.0 -
                                   (CurrentLaserElevationInDegrees -
                                   90.0);
    if (CurrentLaserAzimuthInDegrees > 360.0)
        CurrentLaserAzimuthInDegrees = CurrentLaserAzimuthInDegrees -
                                   360.0;
}
/**************************************************/
/*  THIS IS THE SAME MODULE AS THE OTHER          */
/*  "FindDisplacementAngles" MODULE, EXCEPT THE   */
/*  INPUT PARAMETERS HAVE BEEN ALTERED TO ALLOW A */
/*  SLIGHT PLATFORM POSITION CHANGE FOR THE       */
/*  INTERPOLATION STEPS.  THESE PARAMETERS HAVE   */
/*  BEEN CARRIED OVER TO A SLIGHTLY MODIFIED      */
/*  VERSION OF "TargetPlatform" CALLED            */
/*  "TargetPlatformAgain".  THIS WAS DONE TO AVOID */
/*  ROTATING THE CHANGE IN ECEF POSITION TO A NEW */
/*  LAT AND LON, WHICH WOULD TAKE MORE COMPUTATION */
/*  THAN NECESSARY, AND WOULD DO LITTLE TO CLARIFY */
/*  THE PROBLEM.                                  */
/**************************************************/
```

```
        Dummy = 0.0;
        FindDisplacementAnglesAgain(Platform,
                                    Sat,
                                    ThetaGInRadians,
                                    JulianDate,
                                    ChangeInX,
                                    ChangeInY,
                                    ChangeInZ,
                                    CurrentLaserAzimuthInDegrees,
                                    LaserAzimuthDot,
                                    LaserAzimuthDotDot,
                                    CurrentLaserElevationInDegrees,
                                    LaserElevationDot,
                                    LaserElevationDotDot,
                                    *PlatformSatRENRhoRPtr,
                                    *PlatformSatRENRhoEPtr,
                                    *PlatformSatRENRhoNPtr,
                                    *PlatformSatRENRhoRDotPtr,
                                    *PlatformSatRENRhoEDotPtr,
                                    *PlatformSatRENRhoNDotPtr,
                                    *PlatformSatRENRhoRDotDotPtr,
                                    *PlatformSatRENRhoEDotDotPtr,
                                    *PlatformSatRENRhoNDotDotPtr,
                                    *LaserRENRhoRPtr,
                                    *LaserRENRhoEPtr,
                                    *LaserRENRhoNPtr,
                                    *LaserRENRhoRDotPtr,
                                    *LaserRENRhoEDotPtr,
                                    *LaserRENRhoNDotPtr,
                                    *LaserRENRhoRDotDotPtr,
                                    *LaserRENRhoEDotDotPtr,
                                    *LaserRENRhoNDotDotPtr,
                                    *DummyPtr,
                                    *DummyPtr,
                                    *SeparationAnglePtr,
                                    *DummyPtr,
                                    *DummyPtr,
                                    ErrorList);

        /***************************************************/
        /*  IF THE SATELLITE AND THE LASER ARE GETTING     */
        /*  CLOSER, THEN CONTINUE THE LOOP.  IF THEY BEGIN */
        /*  TO DIVERGE, THEN STOP THE LOOP AND RECORD THE  */
        /*  PREVIOUS SEPARATION ANGLE AS THE CLOSEST       */
        /*  APPROACH ANGLE.                                */
        /***************************************************/
        if (SeparationAngle < LastSepAngle)
        {   Continue = 1;
            StepTime = StepTime + StepInterval;
            LastSepAngle = SeparationAngle;
        }
        else
        {   ClosestApproachInRadians = LastSepAngle;
            TimeToIntersect = TimeElapsed - StepInterval;
            Continue = 0;
        }
    }  /***** END WHILE LOOP ****/

    ClosestApproachInDegrees = ClosestApproachInRadians * RADTODEGREES;

    return;
}
```

```
/*****************************************************************************/
/*  FUNCTION NAME:  TargetPlatformAgain                                      */
/*  AUTHOR:         Captain David Vloedman                                   */
/*  DATE CREATED:   January 24, 1998                                         */
/*                                                                           */
/*  PURPOSE:        This function will take the position of the aircraft and*/
/*                  position,velocity and acceleration in the REN frame of   */
/*                  the Airborn laser platform.  This is very similar to     */
/*                  "TargetPlatform", but uses slightly different input      */
/*                  parameters.                                              */
/*                  NOTICE THAT THIS IS NOT "TargetPlatform", BUT            */
/*                  "TargetPlatformAgain".  IT IS ONLY SLIGHTLY              */
/*                  DIFFERENT THAN THE OTHER, INCORPORATING THE THREE INPUT */
/*                  PARAMETERS ChangeInX, ChangeInY AND ChangeInZ WHICH      */
/*                  DESCRIBES A SLIGHT POSITION CHANGE IN THE ECEF FRAME.    */
/*                                                                           */
/*  INPUTS:         NAME:                   DEFINITION:                      */
/*                                          for the Satellite being studied */
/*                  ABLPlatform             Holds all information about ABL */
/*                                          Platform position/disposition    */
/*                  JulianDate              The time to which the position   */
/*                                          of sat should be propagated to   */
/*                  ChangeInX               This parameter simply describes */
/*                                          change in the ECEF X position    */
/*                                          vector which has occurred after */
/*                                          some given time.  This parameter*/
/*                                          along with the Y an Z are the    */
/*                                          only difference this routine has*/
/*                                          with the other "TargetPlatform" */
/*                                          module.                          */
/*                  ChangeInY               This parameter simply describes */
/*                                          change in the ECEF Y position    */
/*                                          vector which has occurred after */
/*                                          some given time.  This parameter*/
/*                                          along with the X an Z are the    */
/*                                          only difference this routine has*/
/*                                          with the other "TargetPlatform" */
/*                                          module.                          */
/*                  ChangeInZ               This parameter simply describes */
/*                                          change in the ECEF Z position    */
/*                                          vector which has occurred after */
/*                                          some given time.  This parameter*/
/*                                          along with the X an Y are the    */
/*                                          only difference this routine has*/
/*                                          with the other "TargetPlatform" */
/*                                          module.                          */
/*  OUTPUTS:        NAME:                   DESCRIPTION:                     */
/*                  PlatformECIRhoX         X magnitude in ECI frame at Jul */
/*                                          date of X pos vector             */
/*                  PlatformECIRhoY         Y magnitude in ECI frame at Jul */
/*                                          date of Y pos vector             */
/*                  PlatformECIRhoZ         Z magnitude in ECI frame at Jul */
/*                                          date of Z pos vector             */
/*                  PlatformECIRhoXDot      X magnitude in ECI frame at Jul */
/*                                          date of X vel vector             */
/*                  PlatformECIRhoYDot      Y magnitude in ECI frame at Jul */
/*                                          date of Y vel vector             */
/*                  PlatformECIRhoZDot      Z magnitude in ECI frame at Jul */
/*                                          date of Z vel vector             */
/*                  PlatformECIRhoXDotDot   X magnitude in ECI frame at Jul */
/*                                          date of X acc vector             */
/*                  PlatformECIRhoYDotDot   Y magnitude in ECI frame at Jul */
/*                                          date of Y acc vector             */
```

```
/*                  PlatformECIRhoZDotDot   Z magnitude in ECI frame at Jul */
/*                                          date of Z acc vector            */
/*                  PlatformRENRhoR         Radial component in Radial, East*/
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoE         East component in Radial, East  */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoN         North component in Radial, East */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoRDot      Radial Velocity in Radial, East */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoEDot      East velocity in Radial, East   */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoNDot      North velocity in Radial, East  */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoRDotDot   Radial accel. in Radial, East   */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoEDotDot   East accel. in Radial, East     */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoNDotDot   North accel. in Radial, East    */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  ECItoRENMatrixXY        The ECI to REN conversion matrix*/
/*                  ErrorList               The Errors which have occurred  */
/*                                                                          */
/*  COMPILER:       Borland C++ Builder3 Standard version                   */
/*                  This compiler should be used to compile and link.       */
/*                                                                          */
/****************************************************************************/
void TargetPlatformAgain(struct Aircraft &Platform,
                         double &ThetaGInRad,
                         double JulianDate,
                         double ChangeInX,
                         double ChangeInY,
                         double ChangeInZ,
                         double &PlatformECIRhoX,
                         double &PlatformECIRhoY,
                         double &PlatformECIRhoZ,
                         double &PlatformECIRhoXDot,
                         double &PlatformECIRhoYDot,
                         double &PlatformECIRhoZDot,
                         double &PlatformECIRhoXDotDot,
                         double &PlatformECIRhoYDotDot,
                         double &PlatformECIRhoZDotDot,
                         double &PlatformRENRhoR,
                         double &PlatformRENRhoE,
                         double &PlatformRENRhoN,
```

```
                              double &PlatformRENRhoRDot,
                              double &PlatformRENRhoEDot,
                              double &PlatformRENRhoNDot,
                              double &PlatformRENRhoRDotDot,
                              double &PlatformRENRhoEDotDot,
                              double &PlatformRENRhoNDotDot,
                              double &ECItoRENMatrix11,
                              double &ECItoRENMatrix12,
                              double &ECItoRENMatrix13,
                              double &ECItoRENMatrix21,
                              double &ECItoRENMatrix22,
                              double &ECItoRENMatrix23,
                              double &ECItoRENMatrix31,
                              double &ECItoRENMatrix32,
                              double &ECItoRENMatrix33,
                              ErrorStructure   &ErrorList)
{
/*****************************/
/*  DECLARE VARIABLES        */
/*****************************/
    double  Latitude;
    double  Longitude;
    double  LatInRadians;
    double  LonInRadians;
    double  RaircraftECF[3];
    double  VaircraftECF[3];
    double  AircraftRadius;
    double  MagnitudeRaircraftECI;
    double  UnitRaircraftECI[3];
    double  MagnitudeOmegaCrossRac;
    double  OmegaCrossRac[3];
    double  OmegaCrossVac[3];
    double  OmegaCrossOmegaCrossRac[3];
    char    buffer[MAXMESSAGELENGTH] = " ";


/*****************************************************/
/*   ERROR CHECK EACH INPUT PARAMETER                */
/*****************************************************/
    if (Platform.GetAltitude() < 0)
    {   sprintf(buffer,"ABL Platform Altitude is very low -> %d",
                    Platform.GetAltitude());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                            0);
    }
    if ((Platform.GetLatitudeHemisphere() != 0) &&
        (Platform.GetLatitudeHemisphere() != 1))
    {    ErrorList.AddError("TargetSatellite",
                           "Latitude Hemisphere must be north(N) or south(S)",
                            1);
    }
    if (Platform.GetLatitudeDegree() < 0)
    {   sprintf(buffer,"Platform Latitude, %d, must be positive",
                    Platform.GetLatitudeDegree());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                            1);
    }
    if (Platform.GetLatitudeDegree() > 90)
    {   sprintf(buffer,"Platform Latitude, %d, must be less than 90 degrees",
                    Platform.GetLatitudeDegree());
```

```
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Platform.GetLatitudeMinute() < 0)
    {   sprintf(buffer,"Platform Latitude minutes, %d, must be positive",
                Platform.GetLatitudeMinute());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Platform.GetLatitudeMinute() > 60)
    {   sprintf(buffer,"Platform Latitude minutes, %d, must be less than 60",
                Platform.GetLatitudeMinute());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Platform.GetLatitudeSecond() < 0)
    {   sprintf(buffer,"Platform Latitude seconds, %d, must be positive",
                Platform.GetLatitudeSecond());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Platform.GetLatitudeSecond() > 60)
    {   sprintf(buffer,"Platform Latitude seconds, %d, must be less than 60",
                Platform.GetLatitudeSecond());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Platform.GetLongitudeDegree() < 0)
    {   sprintf(buffer,"Platform Longitude Deg, %d, must be positive deg East",
                Platform.GetLongitudeDegree());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Platform.GetLongitudeDegree() > 360)
    {   sprintf(buffer,"Platform Longitude Deg, %d, must be < 360",
                Platform.GetLongitudeDegree());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Platform.GetLongitudeMinute() < 0)
    {   sprintf(buffer,"Platform Longitude Min, %d, must be positive",
                Platform.GetLongitudeMinute());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Platform.GetLongitudeMinute() > 60)
    {   sprintf(buffer,"Platform Longitude Min, %d, must be < 60",
                Platform.GetLongitudeMinute());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Platform.GetLongitudeSecond() < 0)
    {   sprintf(buffer,"Platform Longitude Sec, %d, must be positive",
                Platform.GetLongitudeSecond());
```

```
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
    if ((Platform.GetVelocityX() == 0.0) &&
        (Platform.GetVelocityY() == 0.0) &&
        (Platform.GetVelocityZ() == 0.0))
    {   sprintf(buffer,"Platform is not moving, velocity is zero");
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            0);
    }


/************************************************/
/*   BEGIN CALCULATIONS UNLESS CRITICAL ERROR   */
/************************************************/
    if (ErrorList.CriticalError())
        return;

/************************************************/
/*   INITIALIZE OUTPUT VARIABLES                */
/************************************************/
    PlatformECIRhoX = 0.0;
    PlatformECIRhoY = 0.0;
    PlatformECIRhoZ = 0.0;
    PlatformECIRhoXDot = 0.0;
    PlatformECIRhoYDot = 0.0;
    PlatformECIRhoZDot = 0.0;
    PlatformECIRhoXDotDot = 0.0;
    PlatformECIRhoYDotDot = 0.0;
    PlatformECIRhoZDotDot = 0.0;
    PlatformRENRhoR = 0.0;
    PlatformRENRhoE = 0.0;
    PlatformRENRhoN = 0.0;
    PlatformRENRhoRDot = 0.0;
    PlatformRENRhoEDot = 0.0;
    PlatformRENRhoNDot = 0.0;
    PlatformRENRhoRDotDot = 0.0;
    PlatformRENRhoEDotDot = 0.0;
    PlatformRENRhoNDotDot = 0.0;

/************************************************/
/*   FIND LAT AND LON IN RADIANS                */
/*   NOTE THAT -LAT = SOUTHERN LATITUDE         */
/*   LatitudeHemisphere = "0" = NORTH LAT       */
/*   LatitudeHemisphere = "1" = SOUTH LAT       */
/************************************************/
    Latitude =  (Platform.GetLatitudeDegree()) +
                (Platform.GetLatitudeMinute()/60.0) +
                (Platform.GetLatitudeSecond()/3600.0);
    LatInRadians = Latitude * DEGTORADIANS;
    if (Platform.GetLatitudeHemisphere() == 1)
          LatInRadians = -LatInRadians;

    if (Latitude < -90.0)
    {    ErrorList.AddError("EvaluateEphemeris",
                            "Latitude of platform is more than 90 deg south",
                             1);
    }
    if (Latitude > 90.0)
    {    ErrorList.AddError("EvaluateEphemeris",
                            "Latitude of platform is more than 90 deg north",
```

```
                              1);
    }


    Longitude = (Platform.GetLongitudeDegree()) +
                (Platform.GetLongitudeMinute()/60.0) +
                (Platform.GetLongitudeSecond()/3600.0);
    LonInRadians = Longitude * DEGTORADIANS;
    if (Longitude > 360.0)
    {     ErrorList.AddError("EvaluateEphemeris",
                             "Longitude of platform is > 360 deg",
                              1);
    }

/************************************************/
/*   CONVERT LATITUDE, LONGITUDE AND ALTITUDE   */
/*   POSITION OF THE AIRCRAFT TO A RADIAL VECTOR*/
/*   IN THE EARTH-CENTERED EARTH-FIXED COORD.   */
/*   FRAME                                      */
/*     RaircraftECF[0] = X                      */
/*     RaircraftECF[1] = Y                      */
/*     RaircraftECF[2] = Z                      */
/*   NOTE THAT THIS IS THE ONLY FEW LINES THAT  */
/*   ARE DIFFERENT FROM THE OTHER "Target-      */
/*   Platform".  WE JUST INCORPORATED THE CHANGE*/
/*   IN POSITION..."ChangeInX" AND ETC.         */
/************************************************/
    AircraftRadius = EARTHRADIUS + Platform.GetAltitude();

    RaircraftECF[0] = AircraftRadius *
                      cos(LatInRadians) *
                      cos(LonInRadians) +
                      ChangeInX;
    RaircraftECF[1] = AircraftRadius *
                      cos(LatInRadians) *
                      sin(LonInRadians) +
                      ChangeInY;
    RaircraftECF[2] = AircraftRadius *
                      sin(LatInRadians) +
                      ChangeInZ;

/************************************************/
/*   CONVERT EARTH-CENTERED EARTH-FIXED COORD.  */
/*  FRAME TO EARTH-CENTERED-INERTIAL BY USING   */
/*  THETA-G AS THE ROTATION ANGLE.              */
/*     RaircraftECI[0] = X                      */
/*     RaircraftECI[1] = Y                      */
/*     RaircraftECI[2] = Z                      */
/************************************************/
    PlatformECIRhoX = RaircraftECF[0] * cos(ThetaGInRad) -
                      RaircraftECF[1] * sin(ThetaGInRad);
    PlatformECIRhoY = RaircraftECF[0] * sin(ThetaGInRad) +
                      RaircraftECF[1] * cos(ThetaGInRad);
    PlatformECIRhoZ = RaircraftECF[2];

/************************************************/
/*   CONVERT EARTH-CENTERED EARTH-FIXED COORD.  */
/*  FRAME TO EARTH-CENTERED-INERTIAL BY USING   */
/*  THETA-G AS THE ROTATION ANGLE.  NOTE THAT   */
/*  THIS CAPTURES THE ROTATION OF THE EARTH     */
/*  UNDERNEATH THE PLANE.                       */
/*     VaircraftECI[0] = Xdot                   */
/*     VaircraftECI[1] = Ydot                   */
```

```
/*     VaircraftECI[2] = Zdot                        */
/*  THE UNITS HERE IN THE ECI FRAME ARE:             */
/*      KILOMETERS / SEC                             */
/*  SO WE CONVERT INPUTS TO KM/SEC                   */
/***************************************************/

    VaircraftECF[0] = Platform.GetVelocityX() / 3600;
    VaircraftECF[1] = Platform.GetVelocityY() / 3600;
    VaircraftECF[2] = Platform.GetVelocityZ() / 3600;

    PlatformECIRhoXDot = VaircraftECF[0] * cos(ThetaGInRad) -
                         VaircraftECF[1] * sin(ThetaGInRad) -
                         PlatformECIRhoY * TWOPI/(SECSSIDEREALDAY);
    PlatformECIRhoYDot = VaircraftECF[0] * sin(ThetaGInRad) +
                         VaircraftECF[1] * cos(ThetaGInRad) +
                         PlatformECIRhoX * TWOPI/(SECSSIDEREALDAY);
    PlatformECIRhoZDot = VaircraftECF[2];

/****************************************************/
/*  FIND THE UNIT VECTOR IN THE DIRECTION OF THE */
/*  PLATFORM POSITION VECTOR.  THIS IS USED TO   */
/*  FIND THE MAGNITUDE OF COMPONENTS OF OTHER    */
/*  VECTORS IN THE DIRECTION OF THE PLATFORM     */
/*  POSITION VECTOR.                             */
/****************************************************/
    MagnitudeRaircraftECI = sqrt(pow(PlatformECIRhoX,2) +
                                 pow(PlatformECIRhoY,2) +
                                 pow(PlatformECIRhoZ,2));

    UnitRaircraftECI[0] = PlatformECIRhoX / MagnitudeRaircraftECI;
    UnitRaircraftECI[1] = PlatformECIRhoY / MagnitudeRaircraftECI;
    UnitRaircraftECI[2] = PlatformECIRhoZ / MagnitudeRaircraftECI;

/******************************************************/
/* FIND THE ACCELERATION OF THE AIRCRAFT IN THE     */
/* ECI FRAME                                        */
/* = 2*Omega X Velocity + Omega X (Omega X Position)*/
/* ASSUME PLANE IS FLYING A NON-ACCELERATING COURSE */
/* ON AUTOPILOT.  (Omega = ANGULAR ROTATION OF EARTH*/
/******************************************************/
    OmegaCrossRac[0] = -(TWOPI/(SECSSIDEREALDAY)) * PlatformECIRhoY;
    OmegaCrossRac[1] =  (TWOPI/(SECSSIDEREALDAY)) * PlatformECIRhoX;
    OmegaCrossRac[2] =  0.0;

    OmegaCrossVac[0] = -2.0*(TWOPI/(SECSSIDEREALDAY)) *
                        (VaircraftECF[0] * sin(ThetaGInRad) +
                        VaircraftECF[1] * cos(ThetaGInRad));
    OmegaCrossVac[1] =  2.0*(TWOPI/(SECSSIDEREALDAY)) *
                        (VaircraftECF[0] * cos(ThetaGInRad) -
                        VaircraftECF[1] * sin(ThetaGInRad));
    OmegaCrossVac[2] =  0.0;

    OmegaCrossOmegaCrossRac[0] = -(TWOPI/(SECSSIDEREALDAY)) *
                                 OmegaCrossRac[1];
    OmegaCrossOmegaCrossRac[1] =  (TWOPI/(SECSSIDEREALDAY)) *
                                  OmegaCrossRac[0];
    OmegaCrossOmegaCrossRac[2] =  0.0;

    PlatformECIRhoXDotDot = OmegaCrossVac[0] + OmegaCrossOmegaCrossRac[0];
    PlatformECIRhoYDotDot = OmegaCrossVac[1] + OmegaCrossOmegaCrossRac[1];
    PlatformECIRhoZDotDot = 0.0;
```

```
/**************************************************/
/*  SET UP A CONVERSION MATRIX BETWEEN THE REN    */
/*  ECI COORDINATE FRAMES.                        */
/*  THE REN FRAME IS THE RADIAL, EAST NORTH FRAME*/
/*  WHERE ONE AXIS IS RADIAL UP FROM THE AIRCRAFT*/
/*  OUT OF THE CENTER OF THE EARTH, THE EAST      */
/*  AXIS FOLLOWS THE DIRECTION OF EARTHS ROTATION*/
/*  "EAST" AS VIEWED FROM AIRCRAFT, AND THE NORTH*/
/*  AXIS POINTS TANGENTIALLY TO THE NORTH, AS IT */
/*  WOULD BE SEEN FROM THE AIRCRAFT.              */
/**************************************************/

    MagnitudeOmegaCrossRac = sqrt(pow(OmegaCrossRac[0],2) +
                                  pow(OmegaCrossRac[1],2) +
                                  pow(OmegaCrossRac[2],2));

    ECItoRENMatrix11 = UnitRaircraftECI[0];
    ECItoRENMatrix12 = UnitRaircraftECI[1];
    ECItoRENMatrix13 = UnitRaircraftECI[2];
    ECItoRENMatrix21 = OmegaCrossRac[0] / MagnitudeOmegaCrossRac;
    ECItoRENMatrix22 = OmegaCrossRac[1] / MagnitudeOmegaCrossRac;
    ECItoRENMatrix23 = 0.0;
    ECItoRENMatrix31 = -UnitRaircraftECI[2] *
                       (OmegaCrossRac[1] / MagnitudeOmegaCrossRac);
    ECItoRENMatrix32 = UnitRaircraftECI[2] *
                       (OmegaCrossRac[0] / MagnitudeOmegaCrossRac);
    ECItoRENMatrix33 = (UnitRaircraftECI[0] *
                       (OmegaCrossRac[1] / MagnitudeOmegaCrossRac)) -
                       (UnitRaircraftECI[1] *
                       (OmegaCrossRac[0] / MagnitudeOmegaCrossRac));


/**************************************************/
/*  POSITION VECTOR OF PLATFORM IN THE REN        */
/*  COORDINATE FRAME FROM EARTH CENTER            */
/**************************************************/
    PlatformRENRhoR = ECItoRENMatrix11 * PlatformECIRhoX +
                      ECItoRENMatrix12 * PlatformECIRhoY +
                      ECItoRENMatrix13 * PlatformECIRhoZ;
    PlatformRENRhoE = ECItoRENMatrix21 * PlatformECIRhoX +
                      ECItoRENMatrix22 * PlatformECIRhoY +
                      ECItoRENMatrix23 * PlatformECIRhoZ;
    PlatformRENRhoN = ECItoRENMatrix31 * PlatformECIRhoX +
                      ECItoRENMatrix32 * PlatformECIRhoY +
                      ECItoRENMatrix33 * PlatformECIRhoZ;

/**************************************************/
/*  VELOCITY VECTOR OF PLATFORM IN THE REN        */
/*  COORDINATE FRAME                              */
/**************************************************/
    PlatformRENRhoRDot = ECItoRENMatrix11 * PlatformECIRhoXDot +
                         ECItoRENMatrix12 * PlatformECIRhoYDot +
                         ECItoRENMatrix13 * PlatformECIRhoZDot;
    PlatformRENRhoEDot = ECItoRENMatrix21 * PlatformECIRhoXDot +
                         ECItoRENMatrix22 * PlatformECIRhoYDot +
                         ECItoRENMatrix23 * PlatformECIRhoZDot;
    PlatformRENRhoNDot = ECItoRENMatrix31 * PlatformECIRhoXDot +
                         ECItoRENMatrix32 * PlatformECIRhoYDot +
                         ECItoRENMatrix33 * PlatformECIRhoZDot;

/**************************************************/
/*  ACCELERATION VECTOR OF PLATFORM IN THE REN    */
/*  COORDINATE FRAME                              */
```

```
/**************************************************/
    PlatformRENRhoRDotDot = ECItoRENMatrix11 * PlatformECIRhoXDotDot +
                            ECItoRENMatrix12 * PlatformECIRhoYDotDot +
                            ECItoRENMatrix13 * PlatformECIRhoZDotDot;
    PlatformRENRhoEDotDot = ECItoRENMatrix21 * PlatformECIRhoXDotDot +
                            ECItoRENMatrix22 * PlatformECIRhoYDotDot +
                            ECItoRENMatrix23 * PlatformECIRhoZDotDot;
    PlatformRENRhoNDotDot = ECItoRENMatrix31 * PlatformECIRhoXDotDot +
                            ECItoRENMatrix32 * PlatformECIRhoYDotDot +
                            ECItoRENMatrix33 * PlatformECIRhoZDotDot;

    return;

}

/****************************************************************************/
/*  FUNCTION NAME:  FindDisplacementAnglesAgain                             */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   January 23, 1999                                        */
/*                                                                          */
/*  PURPOSE:        This function will take satellite and platform data and */
/*                  willuse it to find the error angle and the displacement */
/*                  angle between the laser position vector in the REN frame*/
/*                  and the satellite position vector in the same frame.    */
/*                  NOTICE THAT THIS IS NOT "FindDisplacementAngles", BUT   */
/*                  "FindDisplacementAnglesAgain".  IT IS ONLY SLIGHTLY     */
/*                  DIFFERENT THAN THE OTHER, INCORPORATING THE THREE INPUT */
/*                  PARAMETERS ChangeInX, ChangeInY AND ChangeInZ WHICH     */
/*                  DESCRIBES A SLIGHT POSITION CHANGE IN THE ECEF FRAME.   */
/*                                                                          */
/*  INPUTS:         NAME:                  DEFINITION:                      */
/*                  Sat                    Holds all ephemeris information */
/*                                         for the Satellite being studied */
/*                  ABLPlatform            Holds all information about ABL */
/*                                         Platform position/disposition   */
/*                  JulianDate             The time to which the position  */
/*                                         of sat should be propagated to  */
/*                  ChangeInX              This parameter simply describes */
/*                                         change in the ECEF X position   */
/*                                         vector which has occurred after */
/*                                         some given time.  This parameter*/
/*                                         along with the Y an Z are the   */
/*                                         only difference this routine has*/
/*                                         with the other "FindDis..Angles"*/
/*                                         module.                         */
/*                  ChangeInY              This parameter simply describes */
/*                                         change in the ECEF Y position   */
/*                                         vector which has occurred after */
/*                                         some given time.  This parameter*/
/*                                         along with the X an Z are the   */
/*                                         only difference this routine has*/
/*                                         with the other "FindDis..Angles"*/
/*                                         module.                         */
/*                  ChangeInZ              This parameter simply describes */
/*                                         change in the ECEF Z position   */
/*                                         vector which has occurred after */
/*                                         some given time.  This parameter*/
/*                                         along with the X an Y are the   */
/*                                         only difference this routine has*/
/*                                         with the other "FindDis..Angles"*/
/*                                         module.                         */
/*                  ThetaGInRadians        The angle between the Greenwich */
/*                                         Meridian and the Vernal Equinox */
```

```
/*                                              at JulianDate.                  */
/*                  LazerAzimuthInDegrees       Lazer Azimuth at Laze Start time*/
/*                                              in Degrees                      */
/*                  LazerAzimuthDot             The rate of change of the Az    */
/*                                              in Degrees/Sec.                 */
/*                  LazerAzimuthDotDot          The rate of change of the rate  */
/*                                              of change of the Azimuth (Accel)*/
/*                                              in Degrees/Sec^2                */
/*                  LazerElevationInDegrees     Lazer Elevation at Laze Start   */
/*                                              in Degrees                      */
/*                  LazerElevationDot           The rate of change of the El    */
/*                                              in Degrees/Sec.                 */
/*                  LazerElevationDotDot        The rate of change of the rate  */
/*                                              of change of the Elevat. (Accel)*/
/*                                              in Degrees/Sec^2                */
/*  OUTPUTS:        NAME:                       DESCRIPTION:                    */
/*                  PlatformSatRENRhoR          The Radial Component of the     */
/*                                              position vector of the satellite*/
/*                                              wrt the platform in the REN     */
/*                                              coordinate frame.               */
/*                  PlatformSatRENRhoE          The East Component of the       */
/*                                              position vector of the satellite*/
/*                                              wrt the platform in the REN     */
/*                                              coordinate frame.               */
/*                  PlatformSatRENRhoN          The North Component of the      */
/*                                              position vector of the satellite*/
/*                                              wrt the platform in the REN     */
/*                                              coordinate frame.               */
/*                  PlatformSatRENRhoRDot       The Radial Component of the     */
/*                                              velocity vector of the satellite*/
/*                                              wrt the platform in the REN     */
/*                                              coordinate frame.               */
/*                  PlatformSatRENRhoEDot       The East Component of the       */
/*                                              velocity vector of the satellite*/
/*                                              wrt the platform in the REN     */
/*                                              coordinate frame.               */
/*                  PlatformSatRENRhoNDot       The North Component of the      */
/*                                              velocity vector of the satellite*/
/*                                              wrt the platform in the REN     */
/*                                              coordinate frame.               */
/*                  PlatformSatRENRhoRDotDot    The Radial Component of the     */
/*                                              accel vector of the satellite   */
/*                                              wrt the platform in the REN     */
/*                                              coordinate frame.               */
/*                  PlatformSatRENRhoEDotDot    The East Component of the       */
/*                                              accel vector of the satellite   */
/*                                              wrt the platform in the REN     */
/*                                              coordinate frame.               */
/*                  PlatformSatRENRhoNDotDot    The North Component of the      */
/*                                              accel vector of the satellite   */
/*                                              wrt the platform in the REN     */
/*                                              coordinate frame.               */
/*                  LaserRENRhoR                The Radial unit direction of the*/
/*                                              lazer beam trajectory in the REN*/
/*                                              frame.                          */
/*                  LaserRENRhoE                The East unit direction of the  */
/*                                              lazer beam trajectory in the REN*/
/*                                              frame.                          */
/*                  LaserRENRhoN                The North unit direction of the */
/*                                              lazer beam trajectory in the REN*/
/*                                              frame.                          */
/*                  LaserRENRhoRDot             The Radial unit velocity of the */
/*                                              lazer beam trajectory in the REN*/
```

```
/*                                              frame in unit dir*radians/sec    */
/*                   LaserRENRhoEDot            The East unit velocity of the    */
/*                                              lazer beam trajectory in the REN*/
/*                                              frame in unit dir*radians/sec    */
/*                   LaserRENRhoNDot            The North unit velocity of the   */
/*                                              lazer beam trajectory in the REN*/
/*                                              frame in unit dir*radians/sec    */
/*                   LaserRENRhoRDotDot         The Radial unit accel. of the    */
/*                                              lazer beam trajectory in the REN*/
/*                                              frame in unit dir*radians/sec^2 */
/*                   LaserRENRhoEDotDot         The East unit accel. of the      */
/*                                              lazer beam trajectory in the REN*/
/*                                              frame in unit dir*radians/sec^2 */
/*                   LaserRENRhoNDotDot         The North unit accel. of the     */
/*                                              lazer beam trajectory in the REN*/
/*                                              frame in unit dir*radians/sec^2 */
/*                   RangeToSatInKilometers     Holds the range of the aircraft */
/*                                              to the satellite in kilometers. */
/*                   ErrorAngleInRadians        The total error angle in radians*/
/*                   SeparationAngle            The separation (in radians) of  */
/*                                              the LaserRENRho and             */
/*                                              PlatformSatRENRho vectors.      */
/*                   SeparationAngleDot         The rate of change (in rad/sec) */
/*                                              of the separation of LaserRENRho*/
/*                                              PlatformSatRENRho vectors.      */
/*                   SeparationAngleDotDot      The acceleration (in rad/sec^2) */
/*                                              of the separation of LaserRENRho*/
/*                                              and PlatformSatRENRho vectors.  */
/*                   ErrorList                  The Errors which have occurred  */
/*                                                                              */
/*   COMPILER:       Borland C++ Builder3 Standard version                      */
/*                   This compiler should be used to compile and link.          */
/*                                                                              */
/******************************************************************************/
void FindDisplacementAnglesAgain(struct Aircraft &Platform,
                                 struct Satellite &Sat,
                                 double &ThetaGInRad,
                                 double JulianDate,
                                 double ChangeInX,
                                 double ChangeInY,
                                 double ChangeInZ,
                                 double LaserAzimuthInDegrees,
                                 double LaserAzimuthDot,
                                 double LaserAzimuthDotDot,
                                 double LaserElevationInDegrees,
                                 double LaserElevationDot,
                                 double LaserElevationDotDot,
                                 double &PlatformSatRENRhoR,
                                 double &PlatformSatRENRhoE,
                                 double &PlatformSatRENRhoN,
                                 double &PlatformSatRENRhoRDot,
                                 double &PlatformSatRENRhoEDot,
                                 double &PlatformSatRENRhoNDot,
                                 double &PlatformSatRENRhoRDotDot,
                                 double &PlatformSatRENRhoEDotDot,
                                 double &PlatformSatRENRhoNDotDot,
                                 double &LaserRENRhoR,
                                 double &LaserRENRhoE,
                                 double &LaserRENRhoN,
                                 double &LaserRENRhoRDot,
                                 double &LaserRENRhoEDot,
                                 double &LaserRENRhoNDot,
                                 double &LaserRENRhoRDotDot,
```

```
                                double &LaserRENRhoEDotDot,
                                double &LaserRENRhoNDotDot,
                                double &RangeToSatInKilometers,
                                double &ErrorAngleInRadians,
                                double &SeparationAngle,
                                double &SepAngleDot,
                                double &SepAngleDotDot,
                                ErrorStructure   &ErrorList)

{
/*******************************/
/*  VARIABLE DECLARATIONS      */
/*******************************/
    double SatECIRhoX;
    double *SatECIRhoXPtr = &SatECIRhoX;
    double SatECIRhoY;
    double *SatECIRhoYPtr = &SatECIRhoY;
    double SatECIRhoZ;
    double *SatECIRhoZPtr = &SatECIRhoZ;
    double SatECIRhoXDot;
    double *SatECIRhoXDotPtr = &SatECIRhoXDot;
    double SatECIRhoYDot;
    double *SatECIRhoYDotPtr = &SatECIRhoYDot;
    double SatECIRhoZDot;
    double *SatECIRhoZDotPtr = &SatECIRhoZDot;
    double SatECIRhoXDotDot;
    double *SatECIRhoXDotDotPtr = &SatECIRhoXDotDot;
    double SatECIRhoYDotDot;
    double *SatECIRhoYDotDotPtr = &SatECIRhoYDotDot;
    double SatECIRhoZDotDot;
    double *SatECIRhoZDotDotPtr = &SatECIRhoZDotDot;
    double SatRENRhoR;
    double *SatRENRhoRPtr = &SatRENRhoR;
    double SatRENRhoE;
    double *SatRENRhoEPtr = &SatRENRhoE;
    double SatRENRhoN;
    double *SatRENRhoNPtr = &SatRENRhoN;
    double SatRENRhoRDot;
    double *SatRENRhoRDotPtr = &SatRENRhoRDot;
    double SatRENRhoEDot;
    double *SatRENRhoEDotPtr = &SatRENRhoEDot;
    double SatRENRhoNDot;
    double *SatRENRhoNDotPtr = &SatRENRhoNDot;
    double SatRENRhoRDotDot;
    double *SatRENRhoRDotDotPtr = &SatRENRhoRDotDot;
    double SatRENRhoEDotDot;
    double *SatRENRhoEDotDotPtr = &SatRENRhoEDotDot;
    double SatRENRhoNDotDot;
    double *SatRENRhoNDotDotPtr = &SatRENRhoNDotDot;
    double PlatformECIRhoX;
    double *PlatformECIRhoXPtr = &PlatformECIRhoX;
    double PlatformECIRhoY;
    double *PlatformECIRhoYPtr = &PlatformECIRhoY;
    double PlatformECIRhoZ;
    double *PlatformECIRhoZPtr = &PlatformECIRhoZ;
    double PlatformECIRhoXDot;
    double *PlatformECIRhoXDotPtr = &PlatformECIRhoXDot;
    double PlatformECIRhoYDot;
    double *PlatformECIRhoYDotPtr = &PlatformECIRhoYDot;
    double PlatformECIRhoZDot;
    double *PlatformECIRhoZDotPtr = &PlatformECIRhoZDot;
    double PlatformECIRhoXDotDot;
    double *PlatformECIRhoXDotDotPtr = &PlatformECIRhoXDotDot;
```

```
    double PlatformECIRhoYDotDot;
    double *PlatformECIRhoYDotDotPtr = &PlatformECIRhoYDotDot;
    double PlatformECIRhoZDotDot;
    double *PlatformECIRhoZDotDotPtr = &PlatformECIRhoZDotDot;
    double PlatformRENRhoR;
    double *PlatformRENRhoRPtr = &PlatformRENRhoR;
    double PlatformRENRhoE;
    double *PlatformRENRhoEPtr = &PlatformRENRhoE;
    double PlatformRENRhoN;
    double *PlatformRENRhoNPtr = &PlatformRENRhoN;
    double PlatformRENRhoRDot;
    double *PlatformRENRhoRDotPtr = &PlatformRENRhoRDot;
    double PlatformRENRhoEDot;
    double *PlatformRENRhoEDotPtr = &PlatformRENRhoEDot;
    double PlatformRENRhoNDot;
    double *PlatformRENRhoNDotPtr = &PlatformRENRhoNDot;
    double PlatformRENRhoRDotDot;
    double *PlatformRENRhoRDotDotPtr = &PlatformRENRhoRDotDot;
    double PlatformRENRhoEDotDot;
    double *PlatformRENRhoEDotDotPtr = &PlatformRENRhoEDotDot;
    double PlatformRENRhoNDotDot;
    double *PlatformRENRhoNDotDotPtr = &PlatformRENRhoNDotDot;
    double  ECItoRENMatrix11;
    double *ECItoRENMatrix11Ptr = &ECItoRENMatrix11;
    double  ECItoRENMatrix12;
    double *ECItoRENMatrix12Ptr = &ECItoRENMatrix12;
    double  ECItoRENMatrix13;
    double *ECItoRENMatrix13Ptr = &ECItoRENMatrix13;
    double  ECItoRENMatrix21;
    double *ECItoRENMatrix21Ptr = &ECItoRENMatrix21;
    double  ECItoRENMatrix22;
    double *ECItoRENMatrix22Ptr = &ECItoRENMatrix22;
    double  ECItoRENMatrix23;
    double *ECItoRENMatrix23Ptr = &ECItoRENMatrix23;
    double  ECItoRENMatrix31;
    double *ECItoRENMatrix31Ptr = &ECItoRENMatrix31;
    double  ECItoRENMatrix32;
    double *ECItoRENMatrix32Ptr = &ECItoRENMatrix32;
    double  ECItoRENMatrix33;
    double *ECItoRENMatrix33Ptr = &ECItoRENMatrix33;

/****************************************************/
/*   FIND THE PLATFORM POSITION, VELOCITY, AND      */
/*   ACCELERATION IN BOTH THE ECI AND REN           */
/*   COORDINATE FRAMES.  AFTER CONVERSION TO THE    */
/*   REN FRAME, ALSO RETURN THE ECI TO REN CON-     */
/*   VERSION MATRIX TO USE IN OTHER ROTATIONS.      */
/*   NOTICE THAT THIS IS NOT "TargetPlatform", BUT */
/*   "TargetPlatformAgain".  IT IS ONLY SLIGHTLY    */
/*   DIFFERENT THAN THE OTHER, INCORPORATING THE    */
/*   THREE INPUT PARAMETERS ChangeInX, ChangeInY    */
/*   AND ChangeInZ WHICH DESCRIBES A SLIGHT         */
/*   POSITION CHANGE IN THE ECEF FRAME.             */
/****************************************************/
    TargetPlatformAgain(Platform,
                        ThetaGInRad,
                        JulianDate,
                        ChangeInX,
                        ChangeInY,
                        ChangeInZ,
                        *PlatformECIRhoXPtr,
                        *PlatformECIRhoYPtr,
                        *PlatformECIRhoZPtr,
```

```
                    *PlatformECIRhoXDotPtr,
                    *PlatformECIRhoYDotPtr,
                    *PlatformECIRhoZDotPtr,
                    *PlatformECIRhoXDotDotPtr,
                    *PlatformECIRhoYDotDotPtr,
                    *PlatformECIRhoZDotDotPtr,
                    *PlatformRENRhoRPtr,
                    *PlatformRENRhoEPtr,
                    *PlatformRENRhoNPtr,
                    *PlatformRENRhoRDotPtr,
                    *PlatformRENRhoEDotPtr,
                    *PlatformRENRhoNDotPtr,
                    *PlatformRENRhoRDotDotPtr,
                    *PlatformRENRhoEDotDotPtr,
                    *PlatformRENRhoNDotDotPtr,
                    *ECItoRENMatrix11Ptr,        /**********************/
                    *ECItoRENMatrix12Ptr,        /*  ECI TO REN MATRIX */
                    *ECItoRENMatrix13Ptr,        /*  USED TO CONVERT   */
                    *ECItoRENMatrix21Ptr,        /*  FROM ECI TO REN   */
                    *ECItoRENMatrix22Ptr,        /*  COORDINATES.      */
                    *ECItoRENMatrix23Ptr,        /**********************/
                    *ECItoRENMatrix31Ptr,
                    *ECItoRENMatrix32Ptr,
                    *ECItoRENMatrix33Ptr,
                    ErrorList);

/***************************************************/
/*  FIND THE SATELLITE POSITION, VELOCITY AND      */
/*  ACCELERATION IN THE ECI FRAME, THEN USE THE    */
/*  ECI TO REN CON MATRIX TO FIND THE REN VERSION. */
/***************************************************/
    TargetSatellite(Sat,
                    JulianDate,
                    ECItoRENMatrix11,
                    ECItoRENMatrix12,
                    ECItoRENMatrix13,
                    ECItoRENMatrix21,
                    ECItoRENMatrix22,
                    ECItoRENMatrix23,
                    ECItoRENMatrix31,
                    ECItoRENMatrix32,
                    ECItoRENMatrix33,
                    *SatECIRhoXPtr,
                    *SatECIRhoYPtr,
                    *SatECIRhoZPtr,
                    *SatECIRhoXDotPtr,
                    *SatECIRhoYDotPtr,
                    *SatECIRhoZDotPtr,
                    *SatECIRhoXDotDotPtr,
                    *SatECIRhoYDotDotPtr,
                    *SatECIRhoZDotDotPtr,
                    *SatRENRhoRPtr,
                    *SatRENRhoEPtr,
                    *SatRENRhoNPtr,
                    *SatRENRhoRDotPtr,
                    *SatRENRhoEDotPtr,
                    *SatRENRhoNDotPtr,
                    *SatRENRhoRDotDotPtr,
                    *SatRENRhoEDotDotPtr,
                    *SatRENRhoNDotDotPtr,
                    ErrorList);

/***************************************************/
```

```
/*  FIND POSITION, VELOCITY AND ACCELERATION        */
/*  VALUES OF VECTOR GOING FROM PLATFORM TO         */
/*  SATELLITE IN PLATFORM-CENTERED REN FRAME        */
/***************************************************/

/*****************/
/* POSITION      */
/*****************/
    PlatformSatRENRhoR = SatRENRhoR - PlatformRENRhoR;
    PlatformSatRENRhoE = SatRENRhoE - PlatformRENRhoE;
    PlatformSatRENRhoN = SatRENRhoN - PlatformRENRhoN;

/*****************/
/* VELOCITY      */
/*****************/
    PlatformSatRENRhoRDot = SatRENRhoRDot - PlatformRENRhoRDot;
    PlatformSatRENRhoEDot = SatRENRhoEDot - PlatformRENRhoEDot;
    PlatformSatRENRhoNDot = SatRENRhoNDot - PlatformRENRhoNDot;

/*****************/
/* ACCELERATION  */
/*****************/
    PlatformSatRENRhoRDotDot = SatRENRhoRDotDot - PlatformRENRhoRDotDot;
    PlatformSatRENRhoEDotDot = SatRENRhoEDotDot - PlatformRENRhoEDotDot;
    PlatformSatRENRhoNDotDot = SatRENRhoNDotDot - PlatformRENRhoNDotDot;



/****************************************************/
/*   FIND THE VECTOR IN THE REN FRAME ASSOCIATED    */
/*   THE CURRENT AZIMUTH AND ELEVATION.  THE        */
/*   VECTOR RETURNED (LaserRENRho) IS THE UNIT      */
/*   DIRECTION VECTOR POINTING IN THE SAME DIR      */
/*   AS THE AZIMUTH AND ELEVATION.                  */
/****************************************************/
TargetLaser(LaserAzimuthInDegrees,
            LaserElevationInDegrees,
            LaserAzimuthDot,
            LaserElevationDot,
            LaserAzimuthDotDot,
            LaserElevationDotDot,
            LaserRENRhoR,
            LaserRENRhoE,
            LaserRENRhoN,
            LaserRENRhoRDot,
            LaserRENRhoEDot,
            LaserRENRhoNDot,
            LaserRENRhoRDotDot,
            LaserRENRhoEDotDot,
            LaserRENRhoNDotDot,
            ErrorList);

/****************************************************/
/*  FIND THE ANGLE THAT SEPARATES THE SATELLITE     */
/*  POSITION VECTOR AND THE LASER TURRET UNIT       */
/*  DIRECTION VECTOR.                               */
/****************************************************/
    FindSeparationAngle(LaserRENRhoR,
                        LaserRENRhoE,
                        LaserRENRhoN,
                        LaserRENRhoRDot,
                        LaserRENRhoEDot,
```

```
                    LaserRENRhoNDot,
                    LaserRENRhoRDotDot,
                    LaserRENRhoEDotDot,
                    LaserRENRhoNDotDot,
                    PlatformSatRENRhoR,
                    PlatformSatRENRhoE,
                    PlatformSatRENRhoN,
                    PlatformSatRENRhoRDot,
                    PlatformSatRENRhoEDot,
                    PlatformSatRENRhoNDot,
                    PlatformSatRENRhoRDotDot,
                    PlatformSatRENRhoEDotDot,
                    PlatformSatRENRhoNDotDot,
                    SeparationAngle,
                    SepAngleDot,
                    SepAngleDotDot,
                    ErrorList);

    return;

}
```

D.8 Satellite.cpp

```
/*************************************************************************/
/*  MODULE NAME:     Satellite.cpp                                       */
/*  AUTHOR:          Captain David Vloedman                              */
/*  DATE CREATED:    July 26, 1998                                       */
/*                                                                       */
/*  PURPOSE:         This module of code houses the Satellite class object.  */
/*                                                                       */
/*  COMPILER:        Borland C++ Builder3 Standard version               */
/*                   This compiler should be used to compile and link.   */
/*                                                                       */
/*************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "Satellite.h"
#include "LaserConstants.h"
/**********************************/
/* C GENERAL LIBRARIES           */
/**********************************/
#include <stdio.h>
#include <iostream.h>

/****************************************/
/*   CREATE THE SATELLITE CONSTRUCTOR   */
/****************************************/
Satellite::Satellite() :
    EccentricAnomaly(0),
    SemiMajorAxis(0),
    Eccentricity(0),
    MeanAnomaly(0),
    RightAscension(0),
    Inclination(0),
    ArgumentOfPerigee(0),
    TrueAnomaly(0),
    ScalarRadius(0),
    SSCNumber(0),
    RevAtEpoch(0),
    EphemerisType(0),
    ElementSetNumber(0),
    EpochYear(0),
    EpochDay(0),
    RevSquared(0),
    RevCubed(0),
    BStarDrag(0),
    MeanMotion(0)
    {

    }

/****************************************/
/*   CREATE THE SATELLITE DESTRUCTOR    */
/****************************************/

Satellite::~Satellite()
```

```
    {
    }

/***************************************************************************/
/******************   SATELLITE MANIPULATION FUNCTIONS   ******************/
/***************************************************************************/

/****************************************/
/*   SET SEMI-MAJOR AXIS                */
/****************************************/
void Satellite::SetSemiMajorAxis(long double sma)
{   SemiMajorAxis = sma;}

/****************************************/
/*   SET ECCENTRICITY                   */
/****************************************/
void Satellite::SetEccentricity(long double e)
{   Eccentricity = e;}

/****************************************/
/*   SET INCLINATION                    */
/****************************************/
void Satellite::SetInclination(long double i)
{   Inclination = i;}


/****************************************/
/*   SET ARGUMENT OF PERIGEE            */
/****************************************/
void Satellite::SetArgumentOfPerigee(long double ap)
{   ArgumentOfPerigee = ap;}

/****************************************/
/*   SET MEAN ANOMALY                   */
/****************************************/
void Satellite::SetMeanAnomaly(long double ma)
{   MeanAnomaly = ma;}

/****************************************/
/*   SET ECCENTRIC ANOMALY              */
/****************************************/
void Satellite::SetEccentricAnomaly(long double ea)
{   EccentricAnomaly = ea;}

/****************************************/
/*   SET TRUE ANOMALY                   */
/****************************************/
void Satellite::SetTrueAnomaly(long double ta)
{   TrueAnomaly = ta;}

/****************************************/
/*   SET SCALAR RADIUS                  */
/****************************************/
void Satellite::SetScalarRadius(long double sr)
{   ScalarRadius = sr; }

/****************************************/
/*   SET NAME                           */
/****************************************/
void Satellite::SetName(char name[MAXNAMELENGTH])
{   strcpy(Name, name); }

/****************************************/
```

```
/*   SET ELEMENT SET NUMBER              */
/****************************************/
void Satellite::SetElementSetNumber(int esetnum)
{   ElementSetNumber = esetnum;  }

/****************************************/
/*   SET SSC NUMBER                      */
/****************************************/
void Satellite::SetSSCNumber(long int ssc)
{   SSCNumber = ssc;  }

/****************************************/
/*   SET REVOLUTION NUMBER AT EPOCH      */
/****************************************/
void Satellite::SetRevAtEpoch(long int rev)
{   RevAtEpoch = rev;  }

/****************************************/
/*   SET SECURITY CLASSIFICATION         */
/****************************************/
void Satellite::SetSecurityClass(char secclass[CLASSLENGTH+1])
{   strcpy(SecurityClass, secclass);  }

/**********************************************/
/*   SET INTERNATIONAL IDENTIFICATION CODE  */
/**********************************************/
void Satellite::SetInternationalID(char intID[INTNUMLENGTH+1])
{   strcpy(InternationalID, intID);  }

/****************************************/
/*   SET EPHEMERIS TYPE                  */
/****************************************/
void Satellite::SetEphemerisType(int etype)
{   EphemerisType = etype;  }

/****************************************/
/*   SET EPOCH YEAR                      */
/****************************************/
void Satellite::SetEpochYear(int eyear)
{   EpochYear = eyear;  }

/****************************************/
/*   SET EPOCH DAY                       */
/****************************************/
void Satellite::SetEpochDay(long double eday)
{   EpochDay = eday;}

/****************************************/
/*   SET REVOLUTIONS SQUARED             */
/****************************************/
void Satellite::SetRevSquared(long double rev2)
{   RevSquared = rev2;  }

/****************************************/
/*   SET REVOLUTIONS SQUARED             */
/****************************************/
void Satellite::SetRevCubed(long double rev3)
{   RevCubed = rev3;  }

/****************************************/
/*   SET DRAG COEFFICIENT                */
/****************************************/
void Satellite::SetBStarDrag(long double bstar)
```

```
{   BStarDrag = bstar;   }

/****************************************/
/*   SET MEAN MOTION                    */
/****************************************/
void Satellite::SetMeanMotion(long double mm)
{   MeanMotion = mm; }

/****************************************/
/*   SET RIGHT ASCENSION                 */
/****************************************/
void Satellite::SetRightAscension(long double ra)
{   RightAscension = ra;}

/****************************************/
/*   SET TLE BUFFER LINE 1              */
/****************************************/
void Satellite::SetTLELine1(char line[MAXINPUTLINELENGTH])
{   strcpy(TLELine1, line); }

/****************************************/
/*   SET TLE BUFFER LINE 2              */
/****************************************/
void Satellite::SetTLELine2(char line[MAXINPUTLINELENGTH])
{   strcpy(TLELine2, line); }




/****************************************/
/*   GET ECCENTRICITY                   */
/****************************************/
long double Satellite::GetEccentricity()
{  return Eccentricity;  }

/****************************************/
/*   GET RIGHT ASCENSION                */
/****************************************/
long double Satellite::GetRightAscension()
{  return RightAscension; }

/****************************************/
/*   GET INCLINATION                    */
/****************************************/
long double Satellite::GetInclination()
{  return Inclination; }

/****************************************/
/*   GET ARGUMENT OF PERIGEE            */
/****************************************/
long double Satellite::GetArgumentOfPerigee()
{  return ArgumentOfPerigee; }

/****************************************/
/*   GET MEAN ANOMALY                   */
/****************************************/
long double Satellite::GetMeanAnomaly()
{  return  MeanAnomaly;  }

/****************************************/
/*   GET ECCENTRIC ANOMALY              */
/****************************************/
```

```
long double Satellite::GetEccentricAnomaly()
{  return EccentricAnomaly;   }

/****************************************/
/*   GET TRUE ANOMALY                   */
/****************************************/
long double Satellite::GetTrueAnomaly()
{  return TrueAnomaly; }

/****************************************/
/*   GET SCALAR RADIUS                  */
/****************************************/
long double Satellite::GetScalarRadius()
{  return ScalarRadius; }

/****************************************/
/*   GET NAME                           */
/****************************************/
char* Satellite::GetName()
{    return Name; }

/****************************************/
/*   GET REVOLUTION NUMBER AT EPOCH     */
/****************************************/
long int Satellite::GetRevAtEpoch()
{    return RevAtEpoch;   }

/****************************************/
/*   GET SSC NUMBER                     */
/****************************************/
long int Satellite::GetSSCNumber()
{    return SSCNumber;   }

/****************************************/
/*   GET SECURITY CLASSIFICATION        */
/****************************************/
char* Satellite::GetSecurityClass()
{    return SecurityClass;   }

/*********************************************/
/*   GET INTERNATIONAL IDENTIFICATION CODE  */
/*********************************************/
char* Satellite::GetInternationalID()
{    return InternationalID;   }

/****************************************/
/*   GET EPHEMERIS TYPE                 */
/****************************************/
int Satellite::GetEphemerisType()
{    return EphemerisType;   }

/****************************************/
/*   GET ELEMENT SET NUMBER             */
/****************************************/
int Satellite::GetElementSetNumber()
{    return ElementSetNumber;   }

/****************************************/
/*   GET EPOCH YEAR                     */
/****************************************/
int Satellite::GetEpochYear()
{    return EpochYear;   }
```

```
/****************************************/
/*   GET EPOCH DAY                     */
/****************************************/
long double Satellite::GetEpochDay()
{   return EpochDay;   }

/****************************************/
/*   GET REVOLUTIONS SQUARED           */
/****************************************/
long double Satellite::GetRevSquared()
{   return RevSquared;   }

/****************************************/
/*   GET REVOLUTIONS CUBED             */
/****************************************/
long double Satellite::GetRevCubed()
{   return RevCubed;   }

/****************************************/
/*   GET DRAG COEFFICIENT              */
/****************************************/
long double Satellite::GetBStarDrag()
{   return BStarDrag;   }

/****************************************/
/*   GET MEAN MOTION                   */
/****************************************/
long double Satellite::GetMeanMotion()
{   return MeanMotion;   }

/****************************************/
/*   GET TLE BUFFER LINE 1             */
/****************************************/
char* Satellite::GetTLELine1()
{   return TLELine1; }

/****************************************/
/*   GET TLE BUFFER LINE 2             */
/****************************************/
char* Satellite::GetTLELine2()
{   return TLELine2; }
```

D.9 SGP4SupportModules.cpp

```
/*****************************************************************************/
/*  MODULE NAME:      SGP4SupportModules.cpp                                 */
/*  AUTHOR:           Captain David Vloedman                                 */
/*  DATE CREATED:     October 20, 1998                                       */
/*                                                                           */
/*  PURPOSE:          This set of modules supports incorporating "SGP4", a   */
/*                    Satellite position/time propagator developed by        */
/*                    United States Space Command. These modules were        */
/*                    developed for SGP4 Version 3.01C.  They simply serve as */
/*                    an interface between this project and SGP4.            */
/*                                                                           */
/*  COMPILER:         Borland C++ Builder3 Standard version                  */
/*                    This compiler should be used to compile and link.      */
/*                                                                           */
/*****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "SGP4Routines.h"
#include "TimeModules.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "SGP4SupportModules.h"
/**********************************/
/* C STANDARD LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/*****************************************************************************/
/**********************       FUCTIONS        *******************************/
/*****************************************************************************/

/*****************************************************************************/
/*  FUNCTION NAME:  CallSGP4                                                 */
/*  AUTHOR:         Captain David Vloedman                                   */
/*  DATE CREATED:   October 20, 1998                                         */
/*                                                                           */
/*  PURPOSE:        This procedure will take elements already existing       */
/*                  within the Predictive Avoidance Project code and adapt   */
/*                  that information slightly to be used by SGP4 version     */
/*                  3.01.  It will then make a call to SGP4 and return the   */
/*                  results.                                                 */
/*                                                                           */
/*  INPUTS:         NAME:                   DEFINITION:                      */
/*                  Sat                     Holds all ephemeris information */
/*                                          for the Satellite being studied */
/*                  JulianDate              The time to which the position   */
```

```
/*                                                of sat should be propagated to  */
/*  OUTPUTS:            NAME:                     DESCRIPTION:                    */
/*                      X                         X axis pos in ECI frame at Jul  */
/*                                                date                            */
/*                      Y                         Y axis pos in ECI frame at Jul  */
/*                                                date                            */
/*                      Z                         Z axis pos in ECI frame at Jul  */
/*                                                date                            */
/*                      Xdot                      Velocity vector in X direction  */
/*                      Ydot                      Velocity vector in Y direction  */
/*                      Zdot                      Velocity vector in Z direction  */
/*                      Inclination               Inclination at Julian Date      */
/*                      RightAscension            Right Ascension at Julian Date  */
/*                      Eccentricity              Eccentricity at Julian Date     */
/*                      ArgumentOfPerigee         Arg of Perigee at Julian Date   */
/*                      Mean Anomaly              The Mean Anomanly at Julian Date*/
/*                      Delta                     The amount of time in seconds   */
/*                                                that has transpired between the */
/*                                                actual ephemeris measurements   */
/*                                                and the Julian Date propagated  */
/*                      ErrorList                 The Errors which have occurred  */
/*                                                                                */
/*                                                                                */
/*  COMPILER:           Borland C++ Builder3 Standard version                     */
/*                      This compiler should be used to compile and link.         */
/*                                                                                */
/********************************************************************************/
CallSGP4(struct  Satellite &Sat,
            double  JulianDate,
            double  &X,
            double  &Y,
            double  &Z,
            double  &Xdot,
            double  &Ydot,
            double  &Zdot,
            double  &Inclination,
            double  &RightAscension,
            double  &Eccentricity,
            double  &MeanMotion,
            double  &ArgumentOfPerigee,
            double  &MeanAnomaly,
            double  &Delta,
            ErrorStructure   &ErrorList)

{
/*******************************************************/
/*  THE DATA STRUCTURES els21 AND sgp4ret ARE THE      */
/*  STRUCTURES USED TO SEND AND RECEIVE INFORMATION    */
/*  TO SGP4.  THESE ARE DEFINED IN THE SGP4Routines.h*/
/*  FILE.                                              */
/*******************************************************/
    els21   SGP4ElSet;
    sgp4ret ReturnElements;
    double  JulianStart;
    int     ErrorCode;

/***********************************************************/
/*  HERE, WE ARE TRANSFERING ALL OF THE EPHEMERIS DATA     */
/*  FROM THE DATA STRUCTURE USED IN THIS SOFTWARE (Sat OF */
/*  TYPE Satellite) TO THE DATA STRUCTURE TYPE CREATED BY */
/*  THE PROGRAMERS OF SGP4 (els21).  THIS DATA STRUCTURE   */
/*  IS SPECIFIC TO SGP4, AND SO WAS NOT USED THROUGHOUT    */
/*  THIS PROJECT< IN THE EVENT THAT WE WISH TO CHANGE TO   */
```

```
/*  A DIFFERENT PROPAGATOR                                      */
/***************************************************************/
    SGP4ElSet.sn        = Sat.GetSSCNumber();
    strcpy(SGP4ElSet.clas,Sat.GetSecurityClass());
    strcpy(SGP4ElSet.intdes,Sat.GetInternationalID());
    SGP4ElSet.eyear     = Sat.GetEpochYear();
    SGP4ElSet.eday      = double(Sat.GetEpochDay());
    SGP4ElSet.ndot      = double(Sat.GetRevSquared());
    SGP4ElSet.nddot     = double(Sat.GetRevCubed());
    SGP4ElSet.bstar     = double(Sat.GetBStarDrag());
    SGP4ElSet.ephtype   = Sat.GetEphemerisType();
    SGP4ElSet.elnum     = Sat.GetElementSetNumber();
    SGP4ElSet.inc       = double(Sat.GetInclination());
    SGP4ElSet.ra        = double(Sat.GetRightAscension());
    SGP4ElSet.ecc       = double(Sat.GetEccentricity());
    SGP4ElSet.per       = double(Sat.GetArgumentOfPerigee());
    SGP4ElSet.ma        = double(Sat.GetMeanAnomaly());
    SGP4ElSet.n         = double(Sat.GetMeanMotion());
    SGP4ElSet.eprev     = Sat.GetRevAtEpoch();

/***************************************************************/
/*  DETERMINE THE JULIAN DATE EQUIVALENT OF THE START TIME*/
/*  OF THE PROPAGATION.  THIS IS THE TIME RECORDED IN THE */
/*  INPUT FILE AS THE TIME AT WHICH THE EPHEMERIS         */
/*  MEASUREMENTS WERE FIRST TAKEN.                        */
/***************************************************************/
    ConvertCalenderToJulian(Sat.GetEpochYear(),
                            1,
                            1,
                            0,
                            0,
                            0,
                            JulianStart,
                            ErrorList);
    JulianStart = JulianStart + Sat.GetEpochDay();

/***************************************************************/
/*  FIND THE AMOUNT OF TIME TO PROPAGATE THE SATELLITE    */
/*  ORBIT BY SUBSTRACTING THE PROPAGATION JULIAN DATE FROM*/
/*  THE START JULIAN DATE, WHEN THE MEASUREMENTS WERE     */
/*  FIRST RECORDED. DELTA IS IN MINUTES IN SGP4.          */
/***************************************************************/
    Delta = JulianDate - JulianStart;

    Delta = Delta * MINUTESPERDAY;
    if (Delta < 0.0)
    {     ErrorList.AddError("CallSGP4",
                            "There has been a propagation backwards in time",
                             0);
    }

    sgp4prop(1,
             &SGP4ElSet,
             Delta,
             &ReturnElements,
             &ErrorCode);
/***************************************************************/
/*  IF THE ERRORCODE RETURNED FROM SGP4 = 0, THEN AN ERROR*/
/*  HAS OCCURRED.                                         */
/***************************************************************/
    if (ErrorCode == 0)
    {     ErrorList.AddError("CallSGP4",
                            "Error returned from SGP4",
```

```
                               1);
    return 0;
    }

/*********************************************************/
/*  EXTRACT ALL NECESSARY INFORMATION FROM THE OUTPUT     */
/*  STRUCTURE (ReturnElements) OF SGP4.  ALL OUTPUT IS    */
/*  EXPRESSED IN DEGREES, RATHER THAN RADIANS.            */
/*********************************************************/
    X = ReturnElements.x;
    Y = ReturnElements.y;
    Z = ReturnElements.z;
    Xdot = ReturnElements.xdot;
    Ydot = ReturnElements.ydot;
    Zdot = ReturnElements.zdot;
    Inclination = ReturnElements.im * RADTODEGREES;
    RightAscension = ReturnElements.Om * RADTODEGREES;
    Eccentricity = ReturnElements.em;
    MeanMotion = ReturnElements.nm;
    ArgumentOfPerigee = ReturnElements.om * RADTODEGREES;
    MeanAnomaly = ReturnElements.mm * RADTODEGREES;


    return 0;
}
```

D.10 TargetLaser.cpp

```
/*****************************************************************************/
/*  MODULE NAME:     TargetLaser.cpp                                        */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    January 11, 1999                                       */
/*                                                                          */
/*  PURPOSE:         This set of modules supports the processor and are     */
/*                   used to evaluate whether or not the satellite is ever  */
/*                   above the platform horizon.                            */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/*****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "LaserConstants.h"
#include "ErrorStructure.h"
#include "TargetLaser.h"
/**********************************/
/* C STANDARD LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/*****************************************************************************/
/***********************        FUCTIONS        *****************************/
/*****************************************************************************/

/*****************************************************************************/
/*  FUNCTION NAME:  TargetLaser                                             */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   January 3, 1999                                         */
/*                                                                          */
/*  PURPOSE:        This routine finds the unit direction vector of the     */
/*                  laser turret given its reported azimuth and elevation.  */
/*                                                                          */
/*  INPUTS:         NAME:                   DEFINITION:                     */
/*                  Azimuth                 This is the Azimuth (reported in*/
/*                                          degrees east of north) of the   */
/*                                          laser turret.                   */
/*                  Elevation               This is the Elevation (reported */
/*                                          in degrees above horizon) of the*/
/*                                          laser turret.                   */
/*                  AzimuthDot              This is the Azimuth rate of     */
/*                                          change of the laser turret.     */
/*                  AzimuthDot              This is the Elevation rate of   */
/*                                          change of the laser turret.     */
/*                  AzimuthDotDot           This is the Azimuth acceleration*/
/*                                           of the laser turret.           */
```

```
/*                 AzimuthDotDot           This is the Elevation accel.    */
/*                                          of the laser turret.            */
/*                                                                         */
/*   OUTPUTS:      NAME:                   DESCRIPTION:                    */
/*                 LaserRENRhoR            The unit Radial component of the*/
/*                                         position vector given in the    */
/*                                         REN (Radial, East, North) coord */
/*                                         frame which is centered on the  */
/*                                         platform.                       */
/*                 LaserRENRhoE            The unit East component of the*/
/*                                         position vector given in the    */
/*                                         REN (Radial, East, North) coord */
/*                                         frame which is centered on the  */
/*                                         platform.                       */
/*                 LaserRENRhoN            The unit North component of the*/
/*                                         position vector given in the    */
/*                                         REN (Radial, East, North) coord */
/*                                         frame which is centered on the  */
/*                                         platform.                       */
/*                 LaserRENRhoRDot         The unit Radial velocity of the */
/*                                         position vector given in the    */
/*                                         REN (Radial, East, North) coord */
/*                                         frame which is centered on the  */
/*                                         platform.                       */
/*                 LaserRENRhoEDot         The unit East velocity of the   */
/*                                         position vector given in the    */
/*                                         REN (Radial, East, North) coord */
/*                                         frame which is centered on the  */
/*                                         platform.                       */
/*                 LaserRENRhoNDot         The unit North velocity of the  */
/*                                         position vector given in the    */
/*                                         REN (Radial, East, North) coord */
/*                                         frame which is centered on the  */
/*                                         platform.                       */
/*                 LaserRENRhoRDotDot      The unit Radial accel. of the   */
/*                                         position vector given in the    */
/*                                         REN (Radial, East, North) coord */
/*                                         frame which is centered on the  */
/*                                         platform.                       */
/*                 LaserRENRhoEDotDot      The unit East accel. of the     */
/*                                         position vector given in the    */
/*                                         REN (Radial, East, North) coord */
/*                                         frame which is centered on the  */
/*                                         platform.                       */
/*                 LaserRENRhoNDotDot      The unit North accel. of the    */
/*                                         position vector given in the    */
/*                                         REN (Radial, East, North) coord */
/*                                         frame which is centered on the  */
/*                                         platform.                       */
/*                 ErrorList               The Errors which have occurred  */
/*                                                                         */
/*   COMPILER:     Borland C++ Builder3 Standard version                   */
/*                 This compiler should be used to compile and link.       */
/*                                                                         */
/***************************************************************************/
void TargetLaser(double AzimuthInDegrees,
                 double ElevationInDegrees,
                 double AzimuthDot,
                 double ElevationDot,
                 double AzimuthDotDot,
                 double ElevationDotDot,
                 double &LaserRENRhoR,
                 double &LaserRENRhoE,
```

```
                double &LaserRENRhoN,
                double &LaserRENRhoRDot,
                double &LaserRENRhoEDot,
                double &LaserRENRhoNDot,
                double &LaserRENRhoRDotDot,
                double &LaserRENRhoEDotDot,
                double &LaserRENRhoNDotDot,
                ErrorStructure   &ErrorList)
{
/***********************************************/
/*  DECLARE VARIABLES                          */
/***********************************************/
    double  AzimuthInRadians;
    double  ElevationInRadians;
    char    buffer[MAXMESSAGELENGTH] = " ";

/***************************************************/
/*   ERROR CHECK EACH PARAMETER                    */
/***************************************************/
    if (AzimuthInDegrees < 0.0)
    {   sprintf(buffer,"Azimuth cannot be negative. Azimuth = %d",
                AzimuthInDegrees);
        ErrorList.AddError("TargetLaser",
                           buffer,
                           1);
    }
    if (AzimuthInDegrees > 360.0)
    {   sprintf(buffer,"Azimuth should not be > 360. Azimuth = %d",
                AzimuthInDegrees);
        ErrorList.AddError("TargetLaser",
                           buffer,
                           1);
    }
    if (ElevationInDegrees < -90.0)
    {   sprintf(buffer,"Elevation cannot be less than -90 deg. Elevation = %d",
                ElevationInDegrees);
        ErrorList.AddError("TargetLaser",
                           buffer,
                           1);
    }
    if (ElevationInDegrees > 90.0)
    {   sprintf(buffer,"Elevation cannot be > 90 deg. Elevation = %d",
                ElevationInDegrees);
        ErrorList.AddError("TargetLaser",
                           buffer,
                           1);
    }

/*************************************************/
/*   INITIALIZE OUTPUT VARIABLES                 */
/*************************************************/
    LaserRENRhoR = 0.0;
    LaserRENRhoE = 0.0;
    LaserRENRhoN = 0.0;
    LaserRENRhoRDot = 0.0;
    LaserRENRhoEDot = 0.0;
    LaserRENRhoNDot = 0.0;
    LaserRENRhoRDotDot = 0.0;
    LaserRENRhoEDotDot = 0.0;
    LaserRENRhoNDotDot = 0.0;

/*************************************************/
/*   BEGIN CALCULATIONS UNLESS CRITICAL ERROR    */
```

```
/*************************************************/
    if (ErrorList.CriticalError())
        return;

/*************************************************/
/*   CONVERT ALL DEGREE UNITS TO RADIANS         */
/*************************************************/
    ElevationInRadians = ElevationInDegrees * DEGTORADIANS;
    AzimuthInRadians = AzimuthInDegrees * DEGTORADIANS;
    ElevationDot = ElevationDot * DEGTORADIANS;
    AzimuthDot = AzimuthDot * DEGTORADIANS;
    ElevationDotDot = ElevationDotDot * DEGTORADIANS;
    AzimuthDotDot = AzimuthDotDot * DEGTORADIANS;

/*************************************************/
/*   FIND LASER POSITION VECTOR IN REN FRAME     */
/*************************************************/

    LaserRENRhoR = sin(ElevationInRadians);
    LaserRENRhoE = cos(ElevationInRadians) *
                   sin(AzimuthInRadians);
    LaserRENRhoN = cos(ElevationInRadians) *
                   cos(AzimuthInRadians);


/*************************************************/
/*   FIND LASER VELOCITY VECTOR IN REN FRAME     */
/*   THIS IS JUST THE DERIVITIVE OF THE LASER    */
/*   POSITION VECTOR (ABOVE).                    */
/*************************************************/

    LaserRENRhoRDot = cos(ElevationInRadians) * ElevationDot;
    LaserRENRhoEDot = cos(ElevationInRadians) *
                      cos(AzimuthInRadians) * AzimuthDot -
                      sin(ElevationInRadians) *
                      sin(AzimuthInRadians) * ElevationDot;
    LaserRENRhoNDot= -cos(ElevationInRadians) *
                      sin(AzimuthInRadians) * AzimuthDot -
                      sin(ElevationInRadians) *
                      cos(AzimuthInRadians) * ElevationDot;

/*****************************************************/
/*   FIND LASER ACCELERATION VECTOR IN REN FRAME     */
/*   THE ACCELERATION IS JUST THE DERIVITIVE OF THE */
/*   VELOCITY VECTOR DERIVED ABOVE.                  */
/*   NOTE THAT IN ALL THREE OF THESE EQUATIONS:      */
/*   AzimuthInRadians = AZIMUTH IN RADIANS           */
/*   ElevationInRadians = ELEVATION IN RADIANS       */
/*   AzimuthDot = DERIVITIVE OF AZIMUTH              */
/*   ElevationDot = DERIVITIVE OF ELAVATION          */
/*   AzimuthDotDot = ACCELERATION OF AZIMUTH         */
/*   ElevationDotDot = ACCELERATION OF ELEVATION     */
/*****************************************************/
    LaserRENRhoRDotDot = cos(ElevationInRadians) * ElevationDotDot -
                         sin(ElevationInRadians) * ElevationDot * ElevationDot;

    LaserRENRhoEDotDot = cos(ElevationInRadians) *
                         cos(AzimuthInRadians) * AzimuthDotDot -

                         AzimuthDot *
                        (cos(ElevationInRadians) *
                         sin(AzimuthInRadians) * AzimuthDot +
                         sin(ElevationInRadians) *
```

```
                         cos(AzimuthInRadians) * ElevationDot) -

                         sin(ElevationInRadians) *
                         sin(AzimuthInRadians) * ElevationDotDot -

                         ElevationDot *
                        (sin(ElevationInRadians) *
                         cos(AzimuthInRadians) * AzimuthDot +
                         cos(ElevationInRadians) *
                         sin(AzimuthInRadians) * ElevationDot);

    LaserRENRhoNDotDot = -cos(ElevationInRadians) *
                         sin(AzimuthInRadians) * AzimuthDotDot -

                         AzimuthDot *
                        (cos(ElevationInRadians) *
                         cos(AzimuthInRadians) * AzimuthDot -
                         sin(ElevationInRadians) *
                         sin(AzimuthInRadians) * ElevationDot) -

                         sin(ElevationInRadians) *
                         cos(AzimuthInRadians) * ElevationDotDot -

                         ElevationDot *
                        (cos(ElevationInRadians) *
                         cos(AzimuthInRadians) * ElevationDot -
                         sin(ElevationInRadians) *
                         sin(AzimuthInRadians) * AzimuthDot);

    return;

}
```

D.11 TargetPlatform.cpp

```
/****************************************************************************/
/*  MODULE NAME:    TargetPlatform.cpp                                     */
/*  AUTHOR:         Captain David Vloedman                                 */
/*  DATE CREATED:   January 13, 1998                                       */
/*                                                                         */
/*  PURPOSE:        This set of modules supports the processor and are     */
/*                  used to establish the platform's position, velocity, and*/
/*                  acceleration wrt the platform in the REN frame.        */
/*                                                                         */
/*  COMPILER:       Borland C++ Builder3 Standard version                  */
/*                  This compiler should be used to compile and link.      */
/*                                                                         */
/****************************************************************************/
/*********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/*********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/*********************************/
/* USER-BUILT LIBRARIES          */
/*********************************/
#include "LaserConstants.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "TargetPlatform.h"
/*********************************/
/* C STANDARD LIBRARIES          */
/*********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/****************************************************************************/
/**********************         FUCTIONS        ****************************/
/****************************************************************************/

/****************************************************************************/
/*  FUNCTION NAME:  TargetPlatform                                         */
/*  AUTHOR:         Captain David Vloedman                                 */
/*  DATE CREATED:   January 13, 1998                                       */
/*                                                                         */
/*  PURPOSE:        This function will take the position of the aircraft and*/
/*                  position,velocity and acceleration in the REN frame of */
/*                  the Airborn laser platform.                            */
/*                                                                         */
/*  INPUTS:         NAME:                   DEFINITION:                    */
/*                                          for the Satellite being studied */
/*                  ABLPlatform             Holds all information about ABL */
/*                                          Platform position/disposition  */
/*                  JulianDate              The time to which the position */
/*                                          of sat should be propagated to */
/*  OUTPUTS:        NAME:                   DESCRIPTION:                   */
/*                  PlatformECIRhoX         X magnitude in ECI frame at Jul */
/*                                          date of X pos vector           */
/*                  PlatformECIRhoY         Y magnitude in ECI frame at Jul */
/*                                          date of Y pos vector           */
```

```
/*                  PlatformECIRhoZ         Z magnitude in ECI frame at Jul */
/*                                          date of Z pos vector            */
/*                  PlatformECIRhoXDot      X magnitude in ECI frame at Jul */
/*                                          date of X vel vector            */
/*                  PlatformECIRhoYDot      Y magnitude in ECI frame at Jul */
/*                                          date of Y vel vector            */
/*                  PlatformECIRhoZDot      Z magnitude in ECI frame at Jul */
/*                                          date of Z vel vector            */
/*                  PlatformECIRhoXDotDot   X magnitude in ECI frame at Jul */
/*                                          date of X acc vector            */
/*                  PlatformECIRhoYDotDot   Y magnitude in ECI frame at Jul */
/*                                          date of Y acc vector            */
/*                  PlatformECIRhoZDotDot   Z magnitude in ECI frame at Jul */
/*                                          date of Z acc vector            */
/*                  PlatformRENRhoR         Radial component in Radial, East*/
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoE         East component in Radial, East  */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoN         North component in Radial, East */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoRDot      Radial Velocity in Radial, East */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoEDot      East velocity in Radial, East   */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoNDot      North velocity in Radial, East  */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoRDotDot   Radial accel. in Radial, East   */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoEDotDot   East accel. in Radial, East     */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  PlatformRENRhoNDotDot   North accel. in Radial, East    */
/*                                          North coordinate frame of the   */
/*                                          Rho vector descibed above in the*/
/*                                          ECI frame                       */
/*                  ECItoRENMatrixXY        The ECI to REN conversion matrix*/
/*                  ErrorList               The Errors which have occurred  */
/*                                                                          */
/*   COMPILER:      Borland C++ Builder3 Standard version                   */
/*                  This compiler should be used to compile and link.       */
/*                                                                          */
/****************************************************************************/
void TargetPlatform(struct Aircraft &Platform,
                    double &ThetaGInRad,
                    double JulianDate,
                    double &PlatformECIRhoX,
                    double &PlatformECIRhoY,
                    double &PlatformECIRhoZ,
```

```
                     double &PlatformECIRhoXDot,
                     double &PlatformECIRhoYDot,
                     double &PlatformECIRhoZDot,
                     double &PlatformECIRhoXDotDot,
                     double &PlatformECIRhoYDotDot,
                     double &PlatformECIRhoZDotDot,
                     double &PlatformRENRhoR,
                     double &PlatformRENRhoE,
                     double &PlatformRENRhoN,
                     double &PlatformRENRhoRDot,
                     double &PlatformRENRhoEDot,
                     double &PlatformRENRhoNDot,
                     double &PlatformRENRhoRDotDot,
                     double &PlatformRENRhoEDotDot,
                     double &PlatformRENRhoNDotDot,
                     double &ECItoRENMatrix11,
                     double &ECItoRENMatrix12,
                     double &ECItoRENMatrix13,
                     double &ECItoRENMatrix21,
                     double &ECItoRENMatrix22,
                     double &ECItoRENMatrix23,
                     double &ECItoRENMatrix31,
                     double &ECItoRENMatrix32,
                     double &ECItoRENMatrix33,
                     ErrorStructure   &ErrorList)
{
/******************************/
/*  DECLARE VARIABLES         */
/******************************/
    double  Latitude;
    double  Longitude;
    double  LatInRadians;
    double  LonInRadians;
    double  RaircraftECF[3];
    double  VaircraftECF[3];
    double  AircraftRadius;
    double  MagnitudeRaircraftECI;
    double  UnitRaircraftECI[3];
    double  MagnitudeOmegaCrossRac;
    double  OmegaCrossRac[3];
    double  OmegaCrossVac[3];
    double  OmegaCrossOmegaCrossRac[3];
    char    buffer[MAXMESSAGELENGTH] = " ";


/****************************************************/
/*   ERROR CHECK EACH INPUT PARAMETER               */
/****************************************************/
    if (Platform.GetAltitude() < 0)
    {   sprintf(buffer,"ABL Platform Altitude is very low -> %d",
                    Platform.GetAltitude());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                             0);
    }
    if ((Platform.GetLatitudeHemisphere() != 0) &&
        (Platform.GetLatitudeHemisphere() != 1))
    {    ErrorList.AddError("TargetSatellite",
                            "Latitude Hemisphere must be north(N) or south(S)",
                             1);
    }
    if (Platform.GetLatitudeDegree() < 0)
```

```
{   sprintf(buffer,"Platform Latitude, %d, must be positive",
                Platform.GetLatitudeDegree());
    ErrorList.AddError("TargetSatellite",
                       buffer,
                        1);
}
if (Platform.GetLatitudeDegree() > 90)
{   sprintf(buffer,"Platform Latitude, %d, must be less than 90 degrees",
                Platform.GetLatitudeDegree());
    ErrorList.AddError("TargetSatellite",
                       buffer,
                        1);
}
if (Platform.GetLatitudeMinute() < 0)
{   sprintf(buffer,"Platform Latitude minutes, %d, must be positive",
                Platform.GetLatitudeMinute());
    ErrorList.AddError("TargetSatellite",
                       buffer,
                        1);
}
if (Platform.GetLatitudeMinute() > 60)
{   sprintf(buffer,"Platform Latitude minutes, %d, must be less than 60",
                Platform.GetLatitudeMinute());
    ErrorList.AddError("TargetSatellite",
                       buffer,
                        1);
}
if (Platform.GetLatitudeSecond() < 0)
{   sprintf(buffer,"Platform Latitude seconds, %d, must be positive",
                Platform.GetLatitudeSecond());
    ErrorList.AddError("TargetSatellite",
                       buffer,
                       1);
}
if (Platform.GetLatitudeSecond() > 60)
{   sprintf(buffer,"Platform Latitude seconds, %d, must be less than 60",
                Platform.GetLatitudeSecond());
    ErrorList.AddError("TargetSatellite",
                       buffer,
                       1);
}
if (Platform.GetLongitudeDegree() < 0)
{   sprintf(buffer,"Platform Longitude Deg, %d, must be positive deg East",
                Platform.GetLongitudeDegree());
    ErrorList.AddError("TargetSatellite",
                       buffer,
                       1);
}
if (Platform.GetLongitudeDegree() > 360)
{   sprintf(buffer,"Platform Longitude Deg, %d, must be < 360",
                Platform.GetLongitudeDegree());
    ErrorList.AddError("TargetSatellite",
                       buffer,
                       1);
}
if (Platform.GetLongitudeMinute() < 0)
{   sprintf(buffer,"Platform Longitude Min, %d, must be positive",
                Platform.GetLongitudeMinute());
    ErrorList.AddError("TargetSatellite",
                       buffer,
                       1);
}
if (Platform.GetLongitudeMinute() > 60)
```

```
    {   sprintf(buffer,"Platform Longitude Min, %d, must be < 60",
                    Platform.GetLongitudeMinute());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
    if (Platform.GetLongitudeSecond() < 0)
    {   sprintf(buffer,"Platform Longitude Sec, %d, must be positive",
                    Platform.GetLongitudeSecond());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
    if ((Platform.GetVelocityX() == 0.0) &&
        (Platform.GetVelocityY() == 0.0) &&
        (Platform.GetVelocityZ() == 0.0))
    {   sprintf(buffer,"Platform is not moving, velocity is zero");
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            0);
    }


/**************************************************/
/*   BEGIN CALCULATIONS UNLESS CRITICAL ERROR     */
/**************************************************/
    if (ErrorList.CriticalError())
        return;

/**************************************************/
/*   INITIALIZE OUTPUT VARIABLES                  */
/**************************************************/
    PlatformECIRhoX = 0.0;
    PlatformECIRhoY = 0.0;
    PlatformECIRhoZ = 0.0;
    PlatformECIRhoXDot = 0.0;
    PlatformECIRhoYDot = 0.0;
    PlatformECIRhoZDot = 0.0;
    PlatformECIRhoXDotDot = 0.0;
    PlatformECIRhoYDotDot = 0.0;
    PlatformECIRhoZDotDot = 0.0;
    PlatformRENRhoR = 0.0;
    PlatformRENRhoE = 0.0;
    PlatformRENRhoN = 0.0;
    PlatformRENRhoRDot = 0.0;
    PlatformRENRhoEDot = 0.0;
    PlatformRENRhoNDot = 0.0;
    PlatformRENRhoRDotDot = 0.0;
    PlatformRENRhoEDotDot = 0.0;
    PlatformRENRhoNDotDot = 0.0;

/**************************************************/
/*   FIND LAT AND LON IN RADIANS                  */
/*   NOTE THAT -LAT = SOUTHERN LATITUDE           */
/*   LatitudeHemisphere = "0" = NORTH LAT         */
/*   LatitudeHemisphere = "1" = SOUTH LAT         */
/**************************************************/
    Latitude =  (Platform.GetLatitudeDegree()) +
                (Platform.GetLatitudeMinute()/60.0) +
                (Platform.GetLatitudeSecond()/3600.0);
    LatInRadians = Latitude * DEGTORADIANS;
    if (Platform.GetLatitudeHemisphere() == 1)
          LatInRadians = -LatInRadians;
```

```
    if (Latitude < -90.0)
    {    ErrorList.AddError("EvaluateEphemeris",
                            "Latitude of platform is more than 90 deg south",
                             1);
    }
    if (Latitude > 90.0)
    {    ErrorList.AddError("EvaluateEphemeris",
                            "Latitude of platform is more than 90 deg north",
                             1);
    }


    Longitude = (Platform.GetLongitudeDegree()) +
                (Platform.GetLongitudeMinute()/60.0) +
                (Platform.GetLongitudeSecond()/3600.0);
    LonInRadians = Longitude * DEGTORADIANS;
    if (Longitude > 360.0)
    {    ErrorList.AddError("EvaluateEphemeris",
                            "Longitude of platform is > 360 deg",
                             1);
    }

/************************************************/
/*   CONVERT LATITUDE, LONGITUDE AND ALTITUDE   */
/*   POSITION OF THE AIRCRAFT TO A RADIAL VECTOR*/
/*   IN THE EARTH-CENTERED EARTH-FIXED COORD.   */
/*   FRAME                                      */
/*     RaircraftECF[0] = X                      */
/*     RaircraftECF[1] = Y                      */
/*     RaircraftECF[2] = Z                      */
/************************************************/
    AircraftRadius = EARTHRADIUS + Platform.GetAltitude();

    RaircraftECF[0] = AircraftRadius *
                      cos(LatInRadians) *
                      cos(LonInRadians);
    RaircraftECF[1] = AircraftRadius *
                      cos(LatInRadians) *
                      sin(LonInRadians);
    RaircraftECF[2] = AircraftRadius *
                      sin(LatInRadians);

/************************************************/
/*   CONVERT EARTH-CENTERED EARTH-FIXED COORD.  */
/*  FRAME TO EARTH-CENTERED-INERTIAL BY USING   */
/*  THETA-G AS THE ROTATION ANGLE.              */
/*     RaircraftECI[0] = X                      */
/*     RaircraftECI[1] = Y                      */
/*     RaircraftECI[2] = Z                      */
/************************************************/
    PlatformECIRhoX = RaircraftECF[0] * cos(ThetaGInRad) -
                      RaircraftECF[1] * sin(ThetaGInRad);
    PlatformECIRhoY = RaircraftECF[0] * sin(ThetaGInRad) +
                      RaircraftECF[1] * cos(ThetaGInRad);
    PlatformECIRhoZ = RaircraftECF[2];

/************************************************/
/*   CONVERT EARTH-CENTERED EARTH-FIXED COORD.  */
/*  FRAME TO EARTH-CENTERED-INERTIAL BY USING   */
/*  THETA-G AS THE ROTATION ANGLE.  NOTE THAT   */
/*  THIS CAPTURES THE ROTATION OF THE EARTH     */
/*  UNDERNEATH THE PLANE.                       */
```

```
/*     VaircraftECI[0] = Xdot                      */
/*     VaircraftECI[1] = Ydot                      */
/*     VaircraftECI[2] = Zdot                      */
/*  THE UNITS HERE IN THE ECI FRAME ARE:           */
/*      KILOMETERS / SEC                           */
/*  SO WE CONVERT INPUTS TO KM/SEC                 */
/*************************************************/

    VaircraftECF[0] = Platform.GetVelocityX() / 3600;
    VaircraftECF[1] = Platform.GetVelocityY() / 3600;
    VaircraftECF[2] = Platform.GetVelocityZ() / 3600;

    PlatformECIRhoXDot = VaircraftECF[0] * cos(ThetaGInRad) -
                         VaircraftECF[1] * sin(ThetaGInRad) -
                         PlatformECIRhoY * TWOPI/(SECSSIDEREALDAY);
    PlatformECIRhoYDot = VaircraftECF[0] * sin(ThetaGInRad) +
                         VaircraftECF[1] * cos(ThetaGInRad) +
                         PlatformECIRhoX * TWOPI/(SECSSIDEREALDAY);
    PlatformECIRhoZDot = VaircraftECF[2];

/**************************************************/
/*  FIND THE UNIT VECTOR IN THE DIRECTION OF THE */
/*  PLATFORM POSITION VECTOR.  THIS IS USED TO   */
/*  FIND THE MAGNITUDE OF COMPONENTS OF OTHER    */
/*  VECTORS IN THE DIRECTION OF THE PLATFORM     */
/*  POSITION VECTOR.                             */
/**************************************************/
    MagnitudeRaircraftECI = sqrt(pow(PlatformECIRhoX,2) +
                                 pow(PlatformECIRhoY,2) +
                                 pow(PlatformECIRhoZ,2));

    UnitRaircraftECI[0] = PlatformECIRhoX / MagnitudeRaircraftECI;
    UnitRaircraftECI[1] = PlatformECIRhoY / MagnitudeRaircraftECI;
    UnitRaircraftECI[2] = PlatformECIRhoZ / MagnitudeRaircraftECI;

/****************************************************/
/* FIND THE ACCELERATION OF THE AIRCRAFT IN THE     */
/* ECI FRAME                                        */
/* = 2*Omega X Velocity + Omega X (Omega X Position)*/
/* ASSUME PLANE IS FLYING A NON-ACCELERATING COURSE */
/* ON AUTOPILOT.  (Omega = ANGULAR ROTATION OF EARTH*/
/****************************************************/
    OmegaCrossRac[0] = -(TWOPI/(SECSSIDEREALDAY)) * PlatformECIRhoY;
    OmegaCrossRac[1] =  (TWOPI/(SECSSIDEREALDAY)) * PlatformECIRhoX;
    OmegaCrossRac[2] =  0.0;

    OmegaCrossVac[0] = -2.0*(TWOPI/(SECSSIDEREALDAY)) *
                         (VaircraftECF[0] * sin(ThetaGInRad) +
                         VaircraftECF[1] * cos(ThetaGInRad));
    OmegaCrossVac[1] =  2.0*(TWOPI/(SECSSIDEREALDAY)) *
                         (VaircraftECF[0] * cos(ThetaGInRad) -
                         VaircraftECF[1] * sin(ThetaGInRad));
    OmegaCrossVac[2] =  0.0;

    OmegaCrossOmegaCrossRac[0] = -(TWOPI/(SECSSIDEREALDAY)) *
                                  OmegaCrossRac[1];
    OmegaCrossOmegaCrossRac[1] =  (TWOPI/(SECSSIDEREALDAY)) *
                                  OmegaCrossRac[0];
    OmegaCrossOmegaCrossRac[2] =  0.0;

    PlatformECIRhoXDotDot = OmegaCrossVac[0] + OmegaCrossOmegaCrossRac[0];
    PlatformECIRhoYDotDot = OmegaCrossVac[1] + OmegaCrossOmegaCrossRac[1];
    PlatformECIRhoZDotDot = 0.0;
```

```
/**************************************************/
/*  SET UP A CONVERSION MATRIX BETWEEN THE REN    */
/*  ECI COORDINATE FRAMES.                        */
/*  THE REN FRAME IS THE RADIAL, EAST NORTH FRAME*/
/*  WHERE ONE AXIS IS RADIAL UP FROM THE AIRCRAFT*/
/*  OUT OF THE CENTER OF THE EARTH, THE EAST      */
/*  AXIS FOLLOWS THE DIRECTION OF EARTHS ROTATION*/
/*  "EAST" AS VIEWED FROM AIRCRAFT, AND THE NORTH*/
/*  AXIS POINTS TANGENTIALLY TO THE NORTH, AS IT */
/*  WOULD BE SEEN FROM THE AIRCRAFT.              */
/**************************************************/

   MagnitudeOmegaCrossRac = sqrt(pow(OmegaCrossRac[0],2) +
                                 pow(OmegaCrossRac[1],2) +
                                 pow(OmegaCrossRac[2],2));

   ECItoRENMatrix11 = UnitRaircraftECI[0];
   ECItoRENMatrix12 = UnitRaircraftECI[1];
   ECItoRENMatrix13 = UnitRaircraftECI[2];
   ECItoRENMatrix21 = OmegaCrossRac[0] / MagnitudeOmegaCrossRac;
   ECItoRENMatrix22 = OmegaCrossRac[1] / MagnitudeOmegaCrossRac;
   ECItoRENMatrix23 = 0.0;
   ECItoRENMatrix31 = -UnitRaircraftECI[2] *
                      (OmegaCrossRac[1] / MagnitudeOmegaCrossRac);
   ECItoRENMatrix32 = UnitRaircraftECI[2] *
                      (OmegaCrossRac[0] / MagnitudeOmegaCrossRac);
   ECItoRENMatrix33 = (UnitRaircraftECI[0] *
                      (OmegaCrossRac[1] / MagnitudeOmegaCrossRac)) -
                      (UnitRaircraftECI[1] *
                      (OmegaCrossRac[0] / MagnitudeOmegaCrossRac));


/**************************************************/
/*  POSITION VECTOR OF PLATFORM IN THE REN        */
/*  COORDINATE FRAME FROM EARTH CENTER            */
/**************************************************/
   PlatformRENRhoR = ECItoRENMatrix11 * PlatformECIRhoX +
                     ECItoRENMatrix12 * PlatformECIRhoY +
                     ECItoRENMatrix13 * PlatformECIRhoZ;
   PlatformRENRhoE = ECItoRENMatrix21 * PlatformECIRhoX +
                     ECItoRENMatrix22 * PlatformECIRhoY +
                     ECItoRENMatrix23 * PlatformECIRhoZ;
   PlatformRENRhoN = ECItoRENMatrix31 * PlatformECIRhoX +
                     ECItoRENMatrix32 * PlatformECIRhoY +
                     ECItoRENMatrix33 * PlatformECIRhoZ;

/**************************************************/
/*  VELOCITY VECTOR OF PLATFORM IN THE REN        */
/*  COORDINATE FRAME                              */
/**************************************************/
   PlatformRENRhoRDot = ECItoRENMatrix11 * PlatformECIRhoXDot +
                        ECItoRENMatrix12 * PlatformECIRhoYDot +
                        ECItoRENMatrix13 * PlatformECIRhoZDot;
   PlatformRENRhoEDot = ECItoRENMatrix21 * PlatformECIRhoXDot +
                        ECItoRENMatrix22 * PlatformECIRhoYDot +
                        ECItoRENMatrix23 * PlatformECIRhoZDot;
   PlatformRENRhoNDot = ECItoRENMatrix31 * PlatformECIRhoXDot +
                        ECItoRENMatrix32 * PlatformECIRhoYDot +
                        ECItoRENMatrix33 * PlatformECIRhoZDot;

/**************************************************/
```

```
/*  ACCELERATION VECTOR OF PLATFORM IN THE REN   */
/*  COORDINATE FRAME                              */
/*************************************************/
    PlatformRENRhoRDotDot = ECItoRENMatrix11 * PlatformECIRhoXDotDot +
                            ECItoRENMatrix12 * PlatformECIRhoYDotDot +
                            ECItoRENMatrix13 * PlatformECIRhoZDotDot;
    PlatformRENRhoEDotDot = ECItoRENMatrix21 * PlatformECIRhoXDotDot +
                            ECItoRENMatrix22 * PlatformECIRhoYDotDot +
                            ECItoRENMatrix23 * PlatformECIRhoZDotDot;
    PlatformRENRhoNDotDot = ECItoRENMatrix31 * PlatformECIRhoXDotDot +
                            ECItoRENMatrix32 * PlatformECIRhoYDotDot +
                            ECItoRENMatrix33 * PlatformECIRhoZDotDot;


    return;

}
```

D.12 TargetSatellite.cpp

```
/****************************************************************************/
/*  MODULE NAME:     TargetSatellite.cpp                                    */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    November 17, 1998                                      */
/*                                                                          */
/*  PURPOSE:         This set of modules supports the preprocessor and are  */
/*                   used to establish the satellites position, velocity, and*/
/*                   acceleration wrt the platform in the REN frame.        */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "TimeModules.h"
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisModules.h"
#include "SGP4SupportModules.h"
#include "TargetSatellite.h"
/**********************************/
/* C STANDARD LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/****************************************************************************/
/***********************        FUCTIONS        ****************************/
/****************************************************************************/

/****************************************************************************/
/*  FUNCTION NAME:  TargetSatellite                                         */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   November 17, 1998                                       */
/*                                                                          */
/*  PURPOSE:        This function will take the position of the aircraft and*/
/*                  the orbital elements of the satellite and calculate     */
/*                  the azimuth and elevation of the satellite from the     */
/*                  Airborn laser platform.                                 */
/*                                                                          */
/*  INPUTS:         NAME:                   DEFINITION:                     */
/*                  Sat                     Holds all ephemeris information */
/*                                          for the Satellite being studied */
/*                  JulianDate              The time to which the position  */
/*                                          of sat should be propagated to  */
```

```
/*                   ECItoRENMatrix(RowCol)  The ECI to REN conversion matrix*/
/*                                           THIS IS USED TO CONVERT FROM ECI*/
/*                                           COORDINATE FRAME TO THE RADIAL, */
/*                                           EAST, NORTH (REN) FRAME.        */
/*   OUTPUTS:        NAME:                   DESCRIPTION:                    */
/*                   SatECIRhoX              X magnitude in ECI frame at Jul */
/*                                           date of sat radial vector - the */
/*                                           platform radial position vector */
/*                   SatECIRhoY              Y magnitude in ECI frame at Jul */
/*                                           date of sat radial vector - the */
/*                                           platform radial position vector */
/*                   SatECIRhoZ              Z magnitude in ECI frame at Jul */
/*                                           date of sat radial vector - the */
/*                                           platform radial position vector */
/*                   SatECIRhoXDot           X velocity in ECI frame at Jul  */
/*                                           date of sat radial vector - vel */
/*                                           in X axis direction.            */
/*                   SatECIRhoYDot           Y velocity in ECI frame at Jul  */
/*                                           date of sat radial vector - vel */
/*                                           in Y axis direction.            */
/*                   SatECIRhoZDot           Z velocity in ECI frame at Jul  */
/*                                           date of sat radial vector - vel */
/*                                           in Z axis direction.            */
/*                   SatECIRhoXDotDot        X accel. in ECI frame at Jul    */
/*                                           date of sat radial vector - acc.*/
/*                                           in X axis direction.            */
/*                   SatECIRhoYDotDot        Y accel. in ECI frame at Jul    */
/*                                           date of sat radial vector - acc.*/
/*                                           in Y axis direction.            */
/*                   SatECIRhoZDotDot        Z accel. in ECI frame at Jul    */
/*                                           date of sat radial vector - acc.*/
/*                                           in Z axis direction.            */
/*                   SatRENRhoR              The Radial Component of the     */
/*                                           position vector of the satellite*/
/*                                           wrt Earth center in the REN     */
/*                                           coordinate frame.               */
/*                   SatRENRhoE              The East Component of the       */
/*                                           position vector of the satellite*/
/*                                           wrt Earth center in the REN     */
/*                                           coordinate frame.               */
/*                   SatRENRhoN              The North Component of the      */
/*                                           position vector of the satellite*/
/*                                           wrt Earth center in the REN     */
/*                                           coordinate frame.               */
/*                   SatRENRhoRDot           The Radial Component of the     */
/*                                           velocity vector of the satellite*/
/*                                           wrt Earth center in the REN     */
/*                                           coordinate frame.               */
/*                   SatRENRhoEDot           The East Component of the       */
/*                                           velocity vector of the satellite*/
/*                                           wrt Earth center in the REN     */
/*                                           coordinate frame.               */
/*                   SatRENRhoNDot           The North Component of the      */
/*                                           velocity vector of the satellite*/
/*                                           wrt Earth center in the REN     */
/*                                           coordinate frame.               */
/*                   SatRENRhoRDotDot        The Radial Component of the     */
/*                                           accel vector of the satellite   */
/*                                           wrt Earth center in the REN     */
/*                                           coordinate frame.               */
/*                   SatRENRhoEDotDot        The East Component of the       */
/*                                           accel vector of the satellite   */
/*                                           wrt Earth center in the REN     */
```

```
/*                                              coordinate frame.               */
/*                  SatRENRhoNDotDot           The North Component of the      */
/*                                              accel vector of the satellite   */
/*                                              wrt Earth center in the REN     */
/*                                              coordinate frame.               */
/*                  ErrorList                  The Errors which have occurred  */
/*                                                                              */
/*   COMPILER:      Borland C++ Builder3 Standard version                      */
/*                  This compiler should be used to compile and link.          */
/*                                                                              */
/******************************************************************************/
void TargetSatellite(struct Satellite &Sat,
                     double JulianDate,
                     double ECItoRENMatrix11,
                     double ECItoRENMatrix12,
                     double ECItoRENMatrix13,
                     double ECItoRENMatrix21,
                     double ECItoRENMatrix22,
                     double ECItoRENMatrix23,
                     double ECItoRENMatrix31,
                     double ECItoRENMatrix32,
                     double ECItoRENMatrix33,
                     double &SatECIRhoX,
                     double &SatECIRhoY,
                     double &SatECIRhoZ,
                     double &SatECIRhoXDot,
                     double &SatECIRhoYDot,
                     double &SatECIRhoZDot,
                     double &SatECIRhoXDotDot,
                     double &SatECIRhoYDotDot,
                     double &SatECIRhoZDotDot,
                     double &SatRENRhoR,
                     double &SatRENRhoE,
                     double &SatRENRhoN,
                     double &SatRENRhoRDot,
                     double &SatRENRhoEDot,
                     double &SatRENRhoNDot,
                     double &SatRENRhoRDotDot,
                     double &SatRENRhoEDotDot,
                     double &SatRENRhoNDotDot,
                     ErrorStructure   &ErrorList)
{
/************************/
/*  DECLARE VARIABLES   */
/************************/
    double  MagSat;
    char    buffer[MAXMESSAGELENGTH] = " ";

    /************************************************/
    /*  THESE VARIABLES ARE DECLARED ONLY FOR CALL TO */
    /*  SGP4                                          */
    /************************************************/
    double  Inclination;
    double  *InclinationPtr = &Inclination;
    double  RightAscension;
    double  *RightAscensionPtr = &RightAscension;
    double  Eccentricity;
    double  *EccentricityPtr = &Eccentricity;
    double  MeanMotion;
    double  *MeanMotionPtr = &MeanMotion;
    double  ArgumentOfPerigee;
    double  *ArgumentOfPerigeePtr = &ArgumentOfPerigee;
    double  MeanAnomaly;
```

```
    double  *MeanAnomalyPtr = &MeanAnomaly;
    double SatX;
    double  *SatXPtr = &SatX;
    double SatY;
    double  *SatYPtr = &SatY;
    double SatZ;
    double  *SatZPtr = &SatZ;
    double SatXdot;
    double  *SatXdotPtr = &SatXdot;
    double SatYdot;
    double  *SatYdotPtr = &SatYdot;
    double SatZdot;
    double  *SatZdotPtr = &SatZdot;
    double Delta;
    double  *DeltaPtr = &Delta;


/**************************************************/
/*   ERROR CHECK EACH PARAMETER                   */
/**************************************************/
    if (Sat.GetRightAscension() < 0)
    {   sprintf(buffer,"Satellite SSC: %d, has negative Right Ascension",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Sat.GetRightAscension() > 360)
    {   sprintf(buffer,"Satellite SSC: %d, has Right Ascension > 360 deg",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Sat.GetEpochDay() < 0)
    {   sprintf(buffer,"Satellite SSC: %d, has an Epoch Day < 0",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Sat.GetEpochDay() > 366)
    {   sprintf(buffer,"Satellite SSC: %d, has an Epoch Day > 366",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Sat.GetEpochYear() < 1950)
    {   sprintf(buffer,"Satellite SSC: %d, has an Epoch Year < 1950!",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
    if (Sat.GetMeanAnomaly() < 0)
    {   sprintf(buffer,"Satellite SSC: %d, has a Mean Anomaly < 0",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
    }
```

```
    if (Sat.GetMeanAnomaly() > 360)
    {   sprintf(buffer,"Satellite SSC: %d, has a Mean Anomaly > 360 deg",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
    if (Sat.GetInclination() < 0)
    {   sprintf(buffer,"Satellite SSC: %d, has an Inclination < 0",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
    if (Sat.GetInclination() > 180)
    {   sprintf(buffer,"Satellite SSC: %d, has an Inclination > 180 deg",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
    if (Sat.GetEccentricity() < 0)
    {   sprintf(buffer,"Satellite SSC: %d, has an Eccentricity < 0",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
    if (Sat.GetEccentricity() >= 1)
    {   sprintf(buffer,"Satellite SSC: %d, has an Eccentricity > 1.0",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
    if (Sat.GetArgumentOfPerigee() < 0)
    {   sprintf(buffer,"Satellite SSC: %d, has an Argument of Perigee < 0",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
    if (Sat.GetArgumentOfPerigee() > 360)
    {   sprintf(buffer,"Satellite SSC: %d, has an Argument of Per > 360 deg",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
    if (Sat.GetMeanMotion() <= 0)
    {   sprintf(buffer,"Mean Motion <= 0.0 for Satellite SSC: %d",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                            buffer,
                            1);
    }
//    if (Sat.GetRevSquared() <= 0)
//    {   sprintf(buffer,"Revs per day squared <= 0.0 for Satellite SSC: %d",
//                  Sat.GetSSCNumber());
//      ErrorList.AddError("TargetSatellite",
//                          buffer,
//                          1);
//    }
```

```
/************************************************/
/*   BEGIN CALCULATIONS UNLESS CRITICAL ERROR   */
/************************************************/
    if (ErrorList.CriticalError())
        return;

/************************************************/
/*   INITIALIZE OUTPUT VARIABLES                */
/************************************************/
    SatECIRhoX = 0.0;
    SatECIRhoY = 0.0;
    SatECIRhoZ = 0.0;
    SatECIRhoXDot = 0.0;
    SatECIRhoYDot = 0.0;
    SatECIRhoZDot = 0.0;
    SatECIRhoXDotDot = 0.0;
    SatECIRhoYDotDot = 0.0;
    SatECIRhoZDotDot = 0.0;


/*************************************************/
/*  FIND THE POSITION AND VELOCITY VECTORS OF THE*/
/*  SATELLITE FOR THE GIVEN PROPAGATION TIME     */
/*  (WHICH IS STORED IN "JulianDate").           */
/*  NOTE: SGP4 CANNOT HANDLE A PERFECTLY ROUND   */
/*  EPHEMERIS (IE Eccentricity CANNOT EQUAL 0.0  */
/*************************************************/
    if (Sat.GetEccentricity() == 0)
    {   sprintf(buffer,"Satellite SSC: %d, has an Eccent = 0.0, SGP4 Error",
                    Sat.GetSSCNumber());
        ErrorList.AddError("TargetSatellite",
                           buffer,
                           1);
        return;
    }
    CallSGP4(Sat,
             JulianDate,
             *SatXPtr,
             *SatYPtr,
             *SatZPtr,
             *SatXdotPtr,
             *SatYdotPtr,
             *SatZdotPtr,
             *InclinationPtr,
             *RightAscensionPtr,
             *EccentricityPtr,
             *MeanMotionPtr,
             *ArgumentOfPerigeePtr,
             *MeanAnomalyPtr,
             *DeltaPtr,
             ErrorList);

/*************************************************/
/*  HERE, I AM SIMPLY MOVING THE PARAMETERS TO   */
/*  A MATRIX.  THIS COULD HAVE BEEN DONE WITH A  */
/*  LOT OF SHORTCUTS, BUT I DO IT THIS LONG WAY  */
/*  TO ENHANCE READABILITY OF THE PROGRAM AS MUCH*/
/*  AS POSSIBLE.                                 */
/*************************************************/
    SatECIRhoX = SatX;
    SatECIRhoY = SatY;
```

```
    SatECIRhoZ = SatZ;

/************************************************/
/*  VELOCITY VECTOR OF SATEILLITE IN THE REN     */
/*  COORDINATE FRAME.  NOTE THE CONVERSION FROM  */
/*  KM/SEC TO KM/HOUR                            */
/************************************************/
    SatECIRhoXDot = SatXdot;
    SatECIRhoYDot = SatYdot;
    SatECIRhoZDot = SatZdot;


/*************************************************/
/*  ACCELERATION OF SATELLITE IS A FAIRLY STANDARD */
/*  EQUATION:  ACC = -u*r/r^3                     */
/*************************************************/
    MagSat = sqrt(pow(SatECIRhoX,2) +
                  pow(SatECIRhoY,2) +
                  pow(SatECIRhoZ,2));

    SatECIRhoXDotDot = -(MUEARTH)*SatECIRhoX/(pow(MagSat,3));
    SatECIRhoYDotDot = -(MUEARTH)*SatECIRhoY/(pow(MagSat,3));
    SatECIRhoZDotDot = -(MUEARTH)*SatECIRhoZ/(pow(MagSat,3));

/************************************************/
/*  POSITION VECTOR OF SAT IN THE REN            */
/*  COORDINATE FRAME FROM EARTH CENTER           */
/************************************************/
    SatRENRhoR = ECItoRENMatrix11 * SatECIRhoX +
                 ECItoRENMatrix12 * SatECIRhoY +
                 ECItoRENMatrix13 * SatECIRhoZ;
    SatRENRhoE = ECItoRENMatrix21 * SatECIRhoX +
                 ECItoRENMatrix22 * SatECIRhoY +
                 ECItoRENMatrix23 * SatECIRhoZ;
    SatRENRhoN = ECItoRENMatrix31 * SatECIRhoX +
                 ECItoRENMatrix32 * SatECIRhoY +
                 ECItoRENMatrix33 * SatECIRhoZ;

/************************************************/
/*  VELOCITY VECTOR OF PLATFORM IN THE REN       */
/*  COORDINATE FRAME                             */
/************************************************/
    SatRENRhoRDot = ECItoRENMatrix11 * SatECIRhoXDot +
                    ECItoRENMatrix12 * SatECIRhoYDot +
                    ECItoRENMatrix13 * SatECIRhoZDot;
    SatRENRhoEDot = ECItoRENMatrix21 * SatECIRhoXDot +
                    ECItoRENMatrix22 * SatECIRhoYDot +
                    ECItoRENMatrix23 * SatECIRhoZDot;
    SatRENRhoNDot = ECItoRENMatrix31 * SatECIRhoXDot +
                    ECItoRENMatrix32 * SatECIRhoYDot +
                    ECItoRENMatrix33 * SatECIRhoZDot;

/************************************************/
/*  ACCELERATION VECTOR OF PLATFORM IN THE REN   */
/*  COORDINATE FRAME                             */
/************************************************/
    SatRENRhoRDotDot = ECItoRENMatrix11 * SatECIRhoXDotDot +
                       ECItoRENMatrix12 * SatECIRhoYDotDot +
                       ECItoRENMatrix13 * SatECIRhoZDotDot;
    SatRENRhoEDotDot = ECItoRENMatrix21 * SatECIRhoXDotDot +
                       ECItoRENMatrix22 * SatECIRhoYDotDot +
                       ECItoRENMatrix23 * SatECIRhoZDotDot;
    SatRENRhoNDotDot = ECItoRENMatrix31 * SatECIRhoXDotDot +
```

```
                           ECItoRENMatrix32 * SatECIRhoYDotDot +
                           ECItoRENMatrix33 * SatECIRhoZDotDot;

    return;

}
```

D.13 TimeModules.cpp

```
/****************************************************************************/
/*  MODULE NAME:     TimeModules.cpp                                        */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    September 10, 1998                                     */
/*                                                                          */
/*  PURPOSE:         This module of code houses the Time routines which are */
/*                   used to retrieve and manuipulate the times used as     */
/*                   reference times for satellite passing.  The numerical  */
/*                   algorithms were provided by Professor Willian Wiesel,  */
/*                   Air Force Institute of Technology who earlier gleaned  */
/*                   the algorithms from the text, "Numerical Recipes". It  */
/*                   was converted from Fortran to C++ by the author.       */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/**********************************/
/* C STANDARD LIBRARIES           */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
/**********************************/
/* USER-BUILT LIBRARIES           */
/**********************************/
#include "TimeModules.h"
#include "ErrorStructure.h"

/****************************************************************************/
/***********************        FUCTIONS        *****************************/
/****************************************************************************/

/****************************************************************************/
/*  FUNCTION NAME:  ConvertCalenderToJulian                                 */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   September 10, 1998                                      */
/*                                                                          */
/*  PURPOSE:        This function will read in the calender date and return */
/*                  the equivalent modified Julian date.  Note that seconds */
/*                  are accurate to only five decimal places.               */
/*                                                                          */
/*  INPUTS:         NAME:                  DEFINITION:                      */
/*                  CYear                  Holds the calender year          */
/*                  Cmonth                 Holds the Calender month(1 - 12)*/
/*                  CDay                   Holds calender day               */
/*                  CHour                  Holds the calender hour          */
/*                  CMinute                Holds the calender minute        */
/*                  CSecond                Holds the calender second        */
/*                  ErrorList              Holds the Errors found           */
/*                                                                          */
/*  OUTPUTS:        NAME:                  DEFINITION:                      */
```

```
/*                  JulianDate              Holds the Julian equivalent to  */
/*                                              the calender date.          */
/*                                                                          */
/*  COMPILER:       Borland C++ Builder3 Standard version                   */
/*                  This compiler should be used to compile and link.       */
/*                                                                          */
/****************************************************************************/
void ConvertCalenderToJulian(int CYear,
                             int CMonth,
                             int CDay,
                             int CHour,
                             int CMinute,
                             double CSecond,
                             double &JulianDate,
                             ErrorStructure &ErrorList)
{
long int IGreg;
long int IJul;
long int Ick;
int JulianYear;
int JulianMonth;
int Ja;

IGreg = 588829;

if (CYear < 0)
    CYear = CYear + 1;

if (CMonth > 2){
    JulianYear = CYear;
    JulianMonth = CMonth + 1;}
else {
    JulianYear = CYear - 1;
    JulianMonth = CMonth + 13; }

IJul = int(365.25 * JulianYear) + int(30.6001 * JulianMonth) + CDay + 1720995;
Ick  = CDay + 31*(CMonth + 12*CYear);

if(Ick >= IGreg){
    Ja = int(0.01*JulianYear);
    IJul = IJul + 2 - Ja + int(0.25 * Ja);}

IJul = IJul - 2440000;

JulianDate = double(IJul) - 0.50000
                         + double(CHour/24.0)
                         + double(CMinute/1440.0)
                         + double(CSecond/86400.0);


return;

}


/****************************************************************************/
/*  FUNCTION NAME:  ConvertJulianToCalender                                 */
/*  AUTHOR:         Captain David Vloedman                                  */
/*  DATE CREATED:   September 10, 1998                                      */
/*                                                                          */
/*  PURPOSE:        This function will read in the Julian date and return   */
/*                  the equivalent calender date.  Note that seconds        */
/*                  are accurate to only five decimal places.               */
```

```
/*                                                                          */
/*  INPUTS:         NAME:                   DEFINITION:                     */
/*                  JulianDate              Holds the Julian equivalent to  */
/*                                              the calender date.          */
/*                                                                          */
/*  OUTPUTS:        NAME:                   DEFINITION:                     */
/*                  CYear                   Holds the calender year         */
/*                  Cmonth                  Holds the Calender month(1 - 12)*/
/*                  CDay                    Holds calender day              */
/*                  CHour                   Holds the calender hour         */
/*                  CMinute                 Holds the calender minute       */
/*                  CSecond                 Holds the calender second       */
/*                  ErrorList               Holds the Errors found          */
/*                                                                          */
/*  COMPILER:       Borland C++ Builder3 Standard version                   */
/*                  This compiler should be used to compile and link.       */
/*                                                                          */
/****************************************************************************/
void ConvertJulianToCalender(int &CYear,
                             int &CMonth,
                             int &CDay,
                             int &CHour,
                             int &CMinute,
                             double &CSecond,
                             double JulianDate,
                             ErrorStructure &ErrorList)
{
double Fraction;
long int IJul;
long int IGreg;
long int Ja;
long int Jb;
long int Jc;
long int Jd;
long int Je;
long int JAlpha;
IGreg = 2299161;

IJul = int(JulianDate + 0.5) + 2440000;
Fraction = JulianDate + 0.5 - double(IJul - 2440000);

if (IJul >= IGreg) {
    JAlpha = int(((IJul - 1867216) - 0.25)/36524.25);
    Ja = IJul + 1 + JAlpha - int(0.25 * JAlpha);}
else
    Ja = IJul;

Jb = Ja + 1524;
Jc = int(6680.0 + ((Jb - 2439870) - 122.1)/365.25);
Jd = 365 * Jc + int(0.25*Jc);
Je = int((Jb - Jd)/30.6001);
CDay = Jb - Jd - int(30.6001 * Je);

CMonth = Je - 1;
if (CMonth > 12)
    CMonth = CMonth - 12;

CYear = Jc - 4715;
if (CMonth > 2)
    CYear = CYear - 1;
if (CYear <= 0)
    CYear = CYear - 1;
```

```
CHour = int(24.0 * Fraction);
Fraction = Fraction - double(CHour)/24.00;

CMinute = int(1440.0 * Fraction);
Fraction = Fraction - double(CMinute)/1440.0;

CSecond = Fraction * 86400.0;

if (CSecond >= 60.0) {
    CSecond = CSecond - 60;
    CMinute = CMinute + 1;}

if (CMinute >= 60) {
    CMinute = CMinute - 60;
    CHour = CHour + 1;}

if (CHour >= 24) {
    CHour = CHour - 24;
    CDay = CDay + 1; }

return;
}
```

D.14 TLEInput.cpp

```
/*****************************************************************************/
/*  MODULE NAME:     TLEInput.h                                              */
/*  AUTHOR:          Captain David Vloedman                                  */
/*  DATE CREATED:    August 18, 1998                                         */
/*                                                                           */
/*  PURPOSE:         This module of code houses the routines which input the */
/*                   Two Line Element (TLE) sets from an input file.         */
/*                                                                           */
/*  COMPILER:        Borland C++ Builder3 Standard version                   */
/*                   This compiler should be used to compile and link.       */
/*                                                                           */
/*****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "ErrorStructure.h"
/**********************************/
/* C STANDARD LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>

/*****************************************************************************/
/***********************        FUCTIONS        ******************************/
/*****************************************************************************/

/*****************************************************************************/
/*  FUNCTION NAME:   ReadTLEFile                                             */
/*  AUTHOR:          Captain David Vloedman                                  */
/*  DATE CREATED:    August 18, 1998                                         */
/*                                                                           */
/*  PURPOSE:         This function will read in the information contained in */
/*                   an input file holding Two Line Element (TLE) sets.      */
/*                   These TLEs hold the ephemeris data for all of the       */
/*                   satellites we will be covering. It uses the TLE         */
/*                   information to populate a satellite data structure which*/
/*                   is used throughout the program.                         */
/*                                                                           */
/*  INPUTS:          NAME:                  DEFINITION:                      */
/*                   FileName               Holds the name of the Input File*/
/*                                                                           */
/*  OUTPUTS:         NAME:                  DEFINITION:                      */
/*                   SatArray               Returns satellite information    */
/*                   ErrorList              Returns error information        */
/*                                                                           */
/*  COMPILER:        Borland C++ Builder3 Standard version                   */
/*                   This compiler should be used to compile and link.       */
/*                                                                           */
```

```
/*****************************************************************************/
void ReadTLEFile(char FileName[MAXNAMELENGTH],
                 struct SatStructure &SatArray,
                 ErrorStructure &ErrorList)

{
    int              i;
    FILE             *TLEInputFile;
    char             SSCString[SSCLENGTH+1]             = " ";
    char             CardString[CARDLENGTH+1]           = " ";
    char             Classification[CLASSLENGTH+1]      = " ";
    char             IntID[INTNUMLENGTH+1]              = " ";
    char             EYear[EYEARLENGTH+1]               = " ";
    char             EDay[EDAYLENGTH+1]                 = " ";
    char             Rev2[REV2LENGTH+1]                 = " ";
    char             Rev3[REV3LENGTH+1]                 = " ";
    char             RevPower[REVPOWERLENGTH+1]         = " ";
    char             BStar[BSTARLENGTH+1]               = " ";
    char             BStarPower[BPOWERLENGTH+1]         = " ";
    char             EType[ETYPELENGTH+1]               = " ";
    char             ElSet[ELSETLENGTH+1]               = " ";
    char             Inclin[INCLINLENGTH+1]             = " ";
    char             RightAs[RIGHTASLENGTH+1]           = " ";
    char             Ecc[ECCLENGTH+1]                   = " ";
    char             ArgPer[ARGPERLENGTH+1]             = " ";
    char             MeanAn[MEANANLENGTH+1]             = " ";
    char             MeanMo[MEANMOLENGTH+1]             = " ";
    char             EpochRev[EPOCHREVLENGTH+1]         = " ";
    int              CardNumber;
    long int         SSCNumber;
    long int         SSCCheck;
    int              EpochYear;
    int              EphemerisType;
    int              ElSetNumber;
    long double      EpochDay;
    long double      RevSquared;
    long double      RevCubed;
    long double      Rev3Power;
    long double      BStarDrag;
    long double      BPower;
    long double      Multiplier;
    long double      Inclination;
    long double      RightAscension;
    long double      Eccentricity;
    long double      ArgumentOfPerigee;
    long double      MeanAnomaly;
    long double      MeanMotion;
    long int         RevolutionNumber;
    char             buffer[MAXINPUTLINELENGTH]       = " ";
    int              InputLinesRead;
    div_t            LineCheck;

    /****************************************************/
    /*  OPEN THE FILE.  IF THE FILE CANNOT BE OPENED,   */
    /*  REPORT THE ERROR.                               */
    /****************************************************/
    if ((TLEInputFile = fopen(FileName, "r"))==NULL)
    {     ErrorList.AddError("ReadTLEFile",
                             "Cannot open TLE Input File",
                             1);
    }
```

```
InputLinesRead = 0;
SatArray.NumSats = 0;
/******************************************************/
/*  READ THE TLE FILE LINE BY LINE UNTIL THE END OF   */
/*  THE FILE IS REACHED, OR UNLESS THERE IS A CRITICAL*/
/*  ERROR WHICH HAS BEEN ENCOUNTERED.                 */
/******************************************************/
while((ErrorList.CriticalError()== NOERROR) &&
      (fgets(buffer, MAXINPUTLINELENGTH, TLEInputFile) != NULL))
{
    /*****************************************/
    /* COUNT THE LINES READ FROM THE FILE    */
    /*****************************************/
    InputLinesRead = InputLinesRead + 1;

     /*************************************************/
    /* GET THE CARD NUMBER (1 OR 2) OF THE ELEMENT READ */
    /*************************************************/
    CardString[0] = buffer[CARDPOS-1];
    CardNumber = atoi(CardString);

    /***********************************************/
    /* FIND REMAINDER OF LINES READ/2 TO DETERMINE     */
    /* IF WE ARE ON AN EVEN OR ODD NUMBER INPUT LINE   */
    /***********************************************/
    LineCheck = div(InputLinesRead, 2);

    /*****************************************/
    /* COUNT THE LINES READ FROM THE FILE    */
    /*****************************************/
    if (CardNumber == 1)
    {   if (LineCheck.rem != 1)
        /*****************************************/
        /* IF CARD "1" LINE FALLS ON AND EVEN LINE*/
        /* OR CARD "2" FALLS ON AN ODD LINE THEN  */
        /* THERE IS AN ERROR.                     */
        /*****************************************/
        {   ErrorList.AddError("ReadTLEFile",
                               "Input line is out of place. Data corrupt:",
                               0);
            ErrorList.AddError(" ",
                               buffer,
                               0);
        }
        /*****************************************/
        /*  READ THROUGH THE FIELDS OF THE FIRST  */
        /*  CARD LINE AND PULL THE RELEVANT       */
        /*  NUMBERS OUT.  ALL OF THE CONSTANTS    */
        /*  BELOW CAN BE FOUND IN                 */
        /*  "LASERCONSTANTS.H"                    */
        /*****************************************/
        /*****************************************/
        /*  GET SSC NUMBER OF SATELLITE           */
        /*****************************************/
        for (i = 0; i<SSCLENGTH; i++)
            SSCString[i] = buffer[i+SSCPOS-1];
        SSCNumber = atoi(SSCString);

        /*****************************************/
        /*  GET CLASSIFICATION OF SATELLITE DATA  */
        /*****************************************/
        Classification[0] = buffer[CLASSPOS-1];
```

```
/*******************************************/
/*  GET INTERNATIONAL ID OF SATELLITE     */
/*******************************************/
for (i = 0; i<INTNUMLENGTH; i++)
    IntID[i] = buffer[i+INTNUMPOS-1];

/*******************************************/
/*  GET EPOCH YEAR OF DATA RECORDING      */
/*******************************************/
for (i = 0; i<EYEARLENGTH; i++)
    EYear[i] = buffer[i+EYEARPOS-1];
EpochYear = atoi(EYear);

/**************************************************/
/*  YEAR IS GIVEN IN TWO DIGITS --- THIS IS AN     */
/*  ATTEMPT TO CONVERT TO FOUR DIGITS TO BYPASS    */
/*  THE Y2K BUG.  THIS MUST BE CHANGED IN 2040.    */
/**************************************************/
if (EpochYear < 40)
    EpochYear = EpochYear + 2000;
else
    EpochYear = EpochYear + 1900;

/*******************************************/
/*  GET EPOCH DAY OF DATA RECORDING       */
/*******************************************/
for (i = 0; i<EDAYLENGTH; i++)
    EDay[i] = buffer[i+EDAYPOS-1];
EpochDay = atof(EDay);

/*******************************************/
/*  GET NUMBER OF REVOLUTIONS SQUARED AS  */
/*  OF THE EPOCH TIME                     */
/*******************************************/
for (i = 0; i<REV2LENGTH; i++)
    Rev2[i] = buffer[i+REV2POS-1];
RevSquared = atof(Rev2);

/*******************************************/
/*  GET NUMBER OF REVOLUTIONS CUBED AS    */
/*  OF THE EPOCH TIME                     */
/*******************************************/
for (i = 0; i<REV3LENGTH; i++)
    Rev3[i] = buffer[i+REV3POS-1];
RevCubed = atof(Rev3);
for (i = 0; i<REVPOWERLENGTH; i++)
    RevPower[i] = buffer[i+REVPOWERPOS-1];
Rev3Power = atof(RevPower);
Multiplier = pow(10, Rev3Power);
RevCubed = RevCubed * Multiplier;
RevCubed = RevCubed / pow(10, REV3LENGTH-1);

/*******************************************/
/*  GET AIR DRAG COEFFICIENT OF SATELLITE */
/*******************************************/
for (i = 0; i<BSTARLENGTH; i++)
    BStar[i] = buffer[i+BSTARPOS-1];
BStarDrag = atof(BStar);
for (i = 0; i<BPOWERLENGTH; i++)
    BStarPower[i] = buffer[i+BPOWERPOS-1];
BPower = atof(BStarPower);
Multiplier = pow(10, BPower);
BStarDrag = BStarDrag * Multiplier;
```

```
    BStarDrag = BStarDrag / pow(10, BSTARLENGTH-1);

    /*******************************************/
    /*  GET EPHEMERIS TYPE                     */
    /*******************************************/
    for (i = 0; i<ETYPELENGTH; i++)
        EType[i] = buffer[i+ETYPEPOS-1];
    EphemerisType = atoi(EType);

    /*******************************************/
    /*  GET ELEMENT SET NUMBER                 */
    /*******************************************/
    for (i = 0; i<ELSETLENGTH; i++)
        ElSet[i] = buffer[i+ELSETPOS-1];
    ElSetNumber = atoi(ElSet);

    /*******************************************/
    /*  RECORD CARD 1 DATA IN SatArray         */
    /*******************************************/
    SatArray.Sat[SatArray.NumSats].SetTLELine1(buffer);
    SatArray.Sat[SatArray.NumSats].SetSSCNumber(SSCNumber);
    SatArray.Sat[SatArray.NumSats].SetSecurityClass(Classification);
    SatArray.Sat[SatArray.NumSats].SetInternationalID(IntID);
    SatArray.Sat[SatArray.NumSats].SetEpochYear(EpochYear);
    SatArray.Sat[SatArray.NumSats].SetEpochDay(EpochDay);
    SatArray.Sat[SatArray.NumSats].SetRevSquared(RevSquared);
    SatArray.Sat[SatArray.NumSats].SetRevCubed(RevCubed);
    SatArray.Sat[SatArray.NumSats].SetBStarDrag(BStarDrag);
    SatArray.Sat[SatArray.NumSats].SetElementSetNumber(ElSetNumber);
    SatArray.Sat[SatArray.NumSats].SetEphemerisType(EphemerisType);

}
else if(CardNumber == 2)
{
    /*******************************************/
    /*  CHECK SSC NUMBER OF SATELLITE TO MAKE */
    /*  SURE THE DATA IS STILL DESCRIBING THE */
    /*  SAME SATELLITE                         */
    /*******************************************/
    for (i = 0; i<SSCLENGTH; i++)
        SSCString[i] = buffer[i+SSCPOS-1];
    SSCCheck = atoi(SSCString);
    if (SSCNumber != SSCCheck)
    {   ErrorList.AddError(" ReadTLEFile",
                           "Invalid SSC Number in element Record:",
                           0);
        ErrorList.AddError(" ",
                           buffer,
                           0);
    }

    /*******************************************/
    /* IF CARD "1" LINE FALLS ON AND EVEN LINE*/
    /* OR CARD "2" FALLS ON AN ODD LINE THEN  */
    /* THERE IS AN ERROR.                      */
    /* LineCheck.rem IS EITHER 0 OR 1. "rem"  */
    /* STANDS FOR "REMAINDER".                 */
    /*******************************************/
    if (LineCheck.rem != 0)
    {   ErrorList.AddError("ReadTLEFile",
                           "Input line is out of place. Data corrupt:",
                           0);
        ErrorList.AddError(" ",
```

```
                        buffer,
                        0);
}

/*******************************************/
/*  GET INCLINATION OF SATELLITE          */
/*******************************************/
for (i = 0; i<INCLINLENGTH; i++)
    Inclin[i] = buffer[i+INCLINPOS-1];
Inclination = atof(Inclin);

/*******************************************/
/*  GET RIGHT ASCENSION OF SATELLITE      */
/*******************************************/
for (i = 0; i<RIGHTASLENGTH; i++)
    RightAs[i] = buffer[i+RIGHTASPOS-1];
RightAscension = atof(RightAs);

/*******************************************/
/*  GET ECCENTRICITY OF SATELLITE */
/*******************************************/
for (i = 0; i<ECCLENGTH; i++)
    Ecc[i] = buffer[i+ECCPOS-1];
Eccentricity = atof(Ecc);
Eccentricity = Eccentricity / 10000000;

/*******************************************/
/*  GET ARGUMENT OF PERIGEE OF SATELLITE  */
/*******************************************/
for (i = 0; i<ARGPERLENGTH; i++)
    ArgPer[i] = buffer[i+ARGPERPOS-1];
ArgumentOfPerigee = atof(ArgPer);

/*******************************************/
/*  GET MEAN ANOMALY OF SATELLITE         */
/*******************************************/
for (i = 0; i<MEANANLENGTH; i++)
    MeanAn[i] = buffer[i+MEANANPOS-1];
MeanAnomaly = atof(MeanAn);

/*******************************************/
/*  GET MEAN MOTION OF SATELLITE          */
/*******************************************/
for (i = 0; i<MEANMOLENGTH; i++)
    MeanMo[i] = buffer[i+MEANMOPOS-1];
MeanMotion = atof(MeanMo);

/*******************************************/
/*  GET REVOLUTION NUMBER AT EPOCH        */
/*******************************************/
for (i = 0; i<EPOCHREVLENGTH; i++)
    EpochRev[i] = buffer[i+EPOCHREVPOS-1];
RevolutionNumber = atoi(EpochRev);

/*******************************************/
/*  RECORD CARD 2 DATA IN SatArray        */
/*******************************************/
SatArray.Sat[SatArray.NumSats].SetTLELine2(buffer);
SatArray.Sat[SatArray.NumSats].SetInclination(Inclination);
SatArray.Sat[SatArray.NumSats].SetRightAscension(RightAscension);
SatArray.Sat[SatArray.NumSats].SetEccentricity(Eccentricity);

SatArray.Sat[SatArray.NumSats].SetArgumentOfPerigee(ArgumentOfPerigee);
```

```
            SatArray.Sat[SatArray.NumSats].SetMeanAnomaly(MeanAnomaly);
            SatArray.Sat[SatArray.NumSats].SetMeanMotion(MeanMotion);
            SatArray.Sat[SatArray.NumSats].SetRevAtEpoch(RevolutionNumber);
            SatArray.NumSats = SatArray.NumSats + 1;
        }
        else
        {     ErrorList.AddError(" ReadTLEFile",
                                 "Invalid Element Record:",
                                 0);
              ErrorList.AddError(" ",
                                 buffer,
                                 0);
        }
    }
    fclose(TLEInputFile);
}
```

Appendix E.
Test Module Code

E.1 EvaluateEphemerisForm.cpp

```
/****************************************************************************/
/*  MODULE NAME:     EvaluateEphemerisForm.cpp                              */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    October 7, 1998                                        */
/*                                                                          */
/*  PURPOSE:         This is the Form which can be used to test the modules */
/*                   created in EvaluateEphemerisModules.cpp.  This  Form   */
/*                   Takes all the inputs to evaluate a single satellite    */
/*                   ephemeris against a single airborne platform.          */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/****************************************************************************/
/*********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/*********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/*********************************/
/* USER-BUILT LIBRARIES          */
/*********************************/
#include "TimeModules.h"
#include "LaserConstants.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisForm.h"
#include "Aircraft.h"
#include "Satellite.h"
#include "EvaluateEphemerisModules.h"
#include "TLEInput.h"
/*********************************/
/* C SPECIFIC LIBRARIES          */
/*********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>

/*******************************/
/*      CREATE THE FORM        */
/*******************************/
TForm1 *Form1;


//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
```

```
/********************************************************/
/*  THIS PROCURE HANDLES THE BUTTON TO ACTUALLY RUN    */
/*  THE PROCESSING OF THE SINGLE EPHEMERIS             */
/********************************************************/
void __fastcall TForm1::EvaluateEphemerisButtonClick(TObject *Sender)
{

/***************************************************/
/*  VARIABLE LIST                                  */
/***************************************************/
    ErrorStructure ErrorList;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    Satellite* Sat;
        Sat = new Satellite;
    Aircraft* ABLPlatform;
        ABLPlatform = new Aircraft;
    int     SatelliteInView;
    int     *SatInViewPtr = &SatelliteInView;
    int     OrbitInView;
    int     *OrbitInViewPtr = &OrbitInView;
    char    Errors[MAXERRORS][MAXMESSAGELENGTH];
    char    buff[MAXNAMELENGTH];
    double  ReferenceHour;              /****************************/
    double  ReferenceMinute;            /*  THE REFERENT ANGLE      */
    double  ReferenceSecond;            /*  OF THETA G IS KNOWN AS  */
    double  RefModJulianDate;           /*  THE REFERENCE ANGLE     */
    double  ThetaGInRadians;            /****************************/
    double  *ThetaPtr = &ThetaGInRadians;
    double  ThetaGInDegrees;
    int     CalcYear;
    int     CalcMonth;
    int     CalcDay;
    int     CalcHour;
    int     CalcMinute;
    double  CalcSecond;
    int     i;

    double  Inclination;
    double  *InclinationPtr = &Inclination;
    double  RightAscension;
    double  *RightAscensionPtr = &RightAscension;
    double  Eccentricity;
    double  *EccentricityPtr = &Eccentricity;
    double  MeanMotion;
    double  *MeanMotionPtr = &MeanMotion;
    double  ArgumentOfPerigee;
    double  *ArgumentOfPerigeePtr = &ArgumentOfPerigee;
    double  MeanAnomaly;
    double  *MeanAnomalyPtr = &MeanAnomaly;
    double SatX;
    double *SatXPtr = &SatX;
    double SatY;
    double *SatYPtr = &SatY;
    double SatZ;
    double *SatZPtr = &SatZ;
    double SatXdot;
    double *SatXdotPtr = &SatXdot;
    double SatYdot;
    double *SatYdotPtr = &SatYdot;
    double SatZdot;
    double *SatZdotPtr = &SatZdot;
    double Delta;
    double *DeltaPtr = &Delta;
```

```
double JulianDate;
double *JulianDatePtr = &JulianDate;
double TimeToNextRun;
double TimeToRise;
double *TimeToRisePtr = &TimeToRise;
double Dvector;
double *DvectorPtr = &Dvector;
double  CriticalRadius;
double  *CriticalRadiusPtr = &CriticalRadius;
double  SatRadius;
double  *SatRadiusPtr = &SatRadius;


/*********************************************/
/*  GET SATELLITE EPHEMERIS INFORMATION      */
/*********************************************/
Sat->SetSSCNumber(SSCEdit->Text.ToInt());
strcpy(buff,ClassEdit->Text.c_str());
Sat->SetSecurityClass(buff);
strcpy(buff,IntIDEdit->Text.c_str());
Sat->SetInternationalID(buff);
Sat->SetEpochYear(EpochYearEdit->Text.ToInt());
Sat->SetEpochDay(EpochDayEdit->Text.ToDouble());
Sat->SetRevSquared(RevSquaredEdit->Text.ToDouble());
Sat->SetRevCubed(RevCubedEdit->Text.ToDouble());
Sat->SetBStarDrag(BStarEdit->Text.ToDouble());
Sat->SetEphemerisType(EphemerisTypeEdit->Text.ToInt());
Sat->SetElementSetNumber(ElSetEdit->Text.ToInt());
Sat->SetInclination(InclinationEdit->Text.ToDouble());
Sat->SetRightAscension(RightAscensionEdit->Text.ToDouble());
Sat->SetEccentricity(EccentricityEdit->Text.ToDouble());
Sat->SetArgumentOfPerigee(ArgumentOfPerigeeEdit->Text.ToDouble());
Sat->SetMeanAnomaly(MeanAnomalyEdit->Text.ToDouble());
Sat->SetMeanMotion(MeanMotionEdit->Text.ToDouble());
Sat->SetRevAtEpoch(RevNumberEdit->Text.ToInt());

/*********************************************/
/*  GET AIRCRAFT POSITION INFORMATION        */
/*********************************************/
ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
strcpy(buff,HemisphereEdit->Text.c_str());
if ((!(strcmp(buff, "N"))) || (!(strcmp(buff, "n"))))
    ABLPlatform->SetLatitudeHemisphere(0);
else if ((!(strcmp(buff, "S"))) || (!(strcmp(buff, "s"))))
    ABLPlatform->SetLatitudeHemisphere(1);
else
{    ErrorList.AddError("EvaluateEphemerisForm",
                        "Lat Hemisphere must be north(N) or south(S)",
                         1);
}
ABLPlatform->SetLatitudeDegree(LatitudeDegreeEdit->Text.ToDouble());
ABLPlatform->SetLatitudeMinute(LatitudeMinuteEdit->Text.ToDouble());
ABLPlatform->SetLatitudeSecond(LatitudeSecondEdit->Text.ToDouble());
ABLPlatform->SetLongitudeDegree(LongitudeDegreeEdit->Text.ToDouble());
ABLPlatform->SetLongitudeMinute(LongitudeMinuteEdit->Text.ToDouble());
ABLPlatform->SetLongitudeSecond(LongitudeSecondEdit->Text.ToDouble());
ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
ABLPlatform->SetVelocityX(VelocityXEdit->Text.ToDouble());
ABLPlatform->SetVelocityY(VelocityYEdit->Text.ToDouble());
ABLPlatform->SetVelocityZ(VelocityZEdit->Text.ToDouble());

/*********************************************/
/*  GET GREENWICH MERIDIAN REFERENCE         */
```

```
    /*********************************************/
    ReferenceHour = ReferenceHourEdit->Text.ToDouble();
    ReferenceMinute = ReferenceMinuteEdit->Text.ToDouble();
    ReferenceSecond = ReferenceSecondEdit->Text.ToDouble();
    RefModJulianDate = RefModJulianDateEdit->Text.ToDouble();
    TimeToNextRun = SecondsToRunEdit->Text.ToDouble();

    /*********************************************/
    /*   GET CURRENT TIME                        */
    /*********************************************/
    CalcYear = CalcYearEdit->Text.ToInt();
    CalcMonth = CalcMonthEdit->Text.ToInt();
    CalcDay = CalcDayEdit->Text.ToInt();
    CalcHour = CalcHourEdit->Text.ToInt();
    CalcMinute = CalcMinuteEdit->Text.ToInt();
    CalcSecond = CalcSecondEdit->Text.ToDouble();

/****************************************************/
/*  FIND THE CURRENT ANGLE OF THETA G AT THE        */
/*  TIME OF PROPAGATION                             */
/****************************************************/
    ThetaGInRadians = 0;
    FindThetaG(ReferenceHour,
               ReferenceMinute,
               ReferenceSecond,
               RefModJulianDate,
               CalcYear,
               CalcMonth,
               CalcDay,
               CalcHour,
               CalcMinute,
               CalcSecond,
               *ThetaPtr,
               *ErrorPtr);

/****************************************************/
/*  CONVERT THE PROPAGATION TIME TO A JULIAN DATE   */
/*  THAT CAN BE RECOGNIZED BY "EvaluateEphemeris".  */
/****************************************************/
    ConvertCalenderToJulian(CalcYear,
                            CalcMonth,
                            CalcDay,
                            CalcHour,
                            CalcMinute,
                            CalcSecond,
                            *JulianDatePtr,
                            *ErrorPtr);

/****************************************************/
/*  EVALUATE WHETHER OR NOT THE SATELLITE IS        */
/*  CURRENTLY WITHIN VIEW OF THE PLATFORM           */
/****************************************************/
    EvaluateEphemeris(  *Sat,
                        *ABLPlatform,
                        ThetaGInRadians,
                        JulianDate,
                        TimeToNextRun,
                        *SatInViewPtr,
                        *OrbitInViewPtr,
                        *SatXPtr,
                        *SatYPtr,
                        *SatZPtr,
                        *SatXdotPtr,
```

```
                                   *SatYdotPtr,
                                   *SatZdotPtr,
                                   *DeltaPtr,
                                   *InclinationPtr,
                                   *RightAscensionPtr,
                                   *EccentricityPtr,
                                   *MeanMotionPtr,
                                   *ArgumentOfPerigeePtr,
                                   *MeanAnomalyPtr,
                                   *DvectorPtr,
                                   *TimeToRisePtr,
                                   *CriticalRadiusPtr,
                                   *SatRadiusPtr,
                                   *ErrorPtr);


/***************************************************/
/*  OUTPUT THE TEST PARAMETERS WHICH MONITOR THE   */
/*  CALCULATIONS IN "EvaluateEphemeris".           */
/***************************************************/
    XEdit->Text = String(SatX);
    YEdit->Text = String(SatY);
    ZEdit->Text = String(SatZ);
    XdotEdit->Text = String(SatXdot);
    YdotEdit->Text = String(SatYdot);
    ZdotEdit->Text = String(SatZdot);
    DeltaEdit->Text = String(Delta);
    InclinOutEdit->Text = String(Inclination);
    RightAsOutEdit->Text = String(RightAscension);
    EccentricityOutEdit->Text = String(Eccentricity);
    MeanMotionOutEdit->Text = String(MeanMotion);
    ArgOfPerigeeOutEdit->Text = String(ArgumentOfPerigee);
    MeanAnomalyOutEdit->Text = String(MeanAnomaly);
    DvectorEdit->Text = String(Dvector);
    TimeToRiseEdit->Text = String(TimeToRise);
    CriticalRadiusEdit->Text = String(CriticalRadius);
    SatRadiusEdit->Text = String(SatRadius);

    ThetaGInDegrees = ThetaGInRadians * RADTODEGREES;
    ThetaGEdit->Text = String(ThetaGInDegrees);

    if (SatelliteInView == 1)
       SatInRangeEdit->Text = "YES";
    else
       SatInRangeEdit->Text = "NO";

    if (OrbitInView == 1)
       EphemerisInRangeEdit->Text = "YES";
    else
       EphemerisInRangeEdit->Text = "NO";


/*****************************************/
/*  PRINT OUT ALL ERROR MESSAGES         */
/*****************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
```

```
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }


}

/**************************************************/
/*  THIS EVENT HANDLER PROCEDURE HANDLES THE BUTTON*/
/*  THAT CAN LOAD A TEST CASE FROM A FILE FOR LATER*/
/*  EXECUTION                                     */
/**************************************************/
void __fastcall TForm1::FileButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
    SatStructure *SatArray = new SatStructure;

    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int i;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    char FileName[MAXNAMELENGTH] = " ";


/**************************************************/
/*  GET NAME OF FILE TO READ TEST CASE FROM       */
/**************************************************/
    strcpy(FileName,FileEdit->Text.c_str());

/**************************************************/
/*  READ ALL SATELLITES FROM THE FILE, AND USE THE */
/*  FIRST SATELLITE IN THE FILE AS THE TEST CASE   */
/**************************************************/
    ReadTLEFile(FileName,
                *SatArray,
                *ErrorPtr);

/**************************************************/
/*  NOTE THE Sat[0] IS THE FIRST SATELLITE IN THE  */
/*  FILE                                           */
/**************************************************/
    SSCEdit->Text = String(SatArray->Sat[0].GetSSCNumber());
    ClassEdit->Text = String(SatArray->Sat[0].GetSecurityClass());
    IntIDEdit->Text = String(SatArray->Sat[0].GetInternationalID());
    EpochYearEdit->Text = String(SatArray->Sat[0].GetEpochYear());
    EpochDayEdit->Text = String(double(SatArray->Sat[0].GetEpochDay()));
    RevSquaredEdit->Text = String(double(SatArray->Sat[0].GetRevSquared()));
    RevCubedEdit->Text = String(double(SatArray->Sat[0].GetRevCubed()));
    BStarEdit->Text = String(double(SatArray->Sat[0].GetBStarDrag()));
    EphemerisTypeEdit->Text = String(SatArray->Sat[0].GetEphemerisType());
    ElSetEdit->Text = String(SatArray->Sat[0].GetElementSetNumber());
    InclinationEdit->Text = String(double(SatArray->Sat[0].GetInclination()));
    RightAscensionEdit->Text              =              String(double(SatArray-
>Sat[0].GetRightAscension()));
    EccentricityEdit->Text               =               String(double(SatArray-
>Sat[0].GetEccentricity()));
    ArgumentOfPerigeeEdit->Text            =            String(double(SatArray-
>Sat[0].GetArgumentOfPerigee()));
    MeanAnomalyEdit->Text = String(double(SatArray->Sat[0].GetMeanAnomaly()));
    MeanMotionEdit->Text = String(double(SatArray->Sat[0].GetMeanMotion()));
    RevNumberEdit->Text = String(SatArray->Sat[0].GetRevAtEpoch());
```

```
/***************************************************/
/*    DISPLAY ALL ERRORS                           */
/***************************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }

}
```

E.2 FindDisplacementAngleForm.cpp

```
/*****************************************************************************/
/*  MODULE NAME:     FindDisplacementAngleForm.cpp                          */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    January 10 , 1998                                      */
/*                                                                          */
/*  PURPOSE:         This is the Form which can be used to test the modules */
/*                   created in FindDisplacementAngle.cpp.  This form       */
/*                   takes all the inputs to evaluate a single satellite    */
/*                   ephemeris against a single airborne platform, and      */
/*                   determines the separation angle of the satellite pos   */
/*                   vector with respect to the ABL laser beam.             */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/*****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/**********************************/
/* USER-BUILT LIBRARIES           */
/**********************************/
#include "TimeModules.h"
#include "LaserConstants.h"
#include "ErrorStructure.h"
#include "FindDisplacementAngleForm.h"
#include "TargetSatellite.h"
#include "TargetLaser.h"
#include "Aircraft.h"
#include "Satellite.h"
#include "EvaluateEphemerisModules.h"
#include "FindDisplacementAngleModules.h"
#include "TLEInput.h"
/**********************************/
/* C SPECIFIC LIBRARIES           */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>

/*******************************/
/*      CREATE THE FORM       */
/*******************************/
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

/**********************************************/
/*   THIS EVENT HANDLER HANDLES THE EXECUTION  */
/*   AND IS ACTIVATED BY CLICKING ON THE       */
```

```
/*   "FIND SEPARATION ANGLES" BUTTON.           */
/***********************************************/
void __fastcall TForm1::EvaluateButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    Satellite* Sat;
        Sat = new Satellite;
    Aircraft* ABLPlatform;
        ABLPlatform = new Aircraft;
    char    Errors[MAXERRORS][MAXMESSAGELENGTH];
    char    buff[MAXNAMELENGTH];
    double  ReferenceHour;              /****************************/
    double  ReferenceMinute;            /*  THE REFERENT ANGLE      */
    double  ReferenceSecond;            /*  OF THETA G IS KNOWN AS  */
    double  RefModJulianDate;           /*  THE REFERENCE ANGLE IN  */
    double  ThetaGInRadians;            /****************************/
    double  *ThetaPtr = &ThetaGInRadians;
    double  ErrorAngleInRadians;
    double  *ErrorAngleInRadiansPtr = &ErrorAngleInRadians;
    double  LaserAzimuthInDegrees;
    double  LaserAzimuthDot;
    double  LaserAzimuthDotDot;
    double  LaserElevationInDegrees;
    double  LaserElevationDot;
    double  LaserElevationDotDot;
    double  SatPositionErrorInMeters;
    double  PlatformPositionErrorInMeters;
    double  MissilePositionErrorInMeters;
    double  RangeToMissileInKilometers;
    double  OtherErrorAngleInDeg;
    int     CalcYear;
    int     CalcMonth;
    int     CalcDay;
    int     CalcHour;
    int     CalcMinute;
    double  CalcSecond;
    int     i;
    double JulianDate;
    double *JulianDatePtr = &JulianDate;
    double RangeToSatInKilometers;
    double *RangeToSatInKilometersPtr = &RangeToSatInKilometers;
    double PlatformSatRENRhoR;
    double *PlatformSatRENRhoRPtr = &PlatformSatRENRhoR;
    double PlatformSatRENRhoE;
    double *PlatformSatRENRhoEPtr = &PlatformSatRENRhoE;
    double PlatformSatRENRhoN;
    double *PlatformSatRENRhoNPtr = &PlatformSatRENRhoN;
    double PlatformSatRENRhoRDot;
    double *PlatformSatRENRhoRDotPtr = &PlatformSatRENRhoRDot;
    double PlatformSatRENRhoEDot;
    double *PlatformSatRENRhoEDotPtr = &PlatformSatRENRhoEDot;
    double PlatformSatRENRhoNDot;
    double *PlatformSatRENRhoNDotPtr = &PlatformSatRENRhoNDot;
    double PlatformSatRENRhoRDotDot;
    double *PlatformSatRENRhoRDotDotPtr = &PlatformSatRENRhoRDotDot;
    double PlatformSatRENRhoEDotDot;
    double *PlatformSatRENRhoEDotDotPtr = &PlatformSatRENRhoEDotDot;
    double PlatformSatRENRhoNDotDot;
    double *PlatformSatRENRhoNDotDotPtr = &PlatformSatRENRhoNDotDot;
    double LaserRENRhoR;
    double *LaserRENRhoRPtr = &LaserRENRhoR;
    double LaserRENRhoE;
```

```
double *LaserRENRhoEPtr = &LaserRENRhoE;
double LaserRENRhoN;
double *LaserRENRhoNPtr = &LaserRENRhoN;
double LaserRENRhoRDot;
double *LaserRENRhoRDotPtr = &LaserRENRhoRDot;
double LaserRENRhoEDot;
double *LaserRENRhoEDotPtr = &LaserRENRhoEDot;
double LaserRENRhoNDot;
double *LaserRENRhoNDotPtr = &LaserRENRhoNDot;
double LaserRENRhoRDotDot;
double *LaserRENRhoRDotDotPtr = &LaserRENRhoRDotDot;
double LaserRENRhoEDotDot;
double *LaserRENRhoEDotDotPtr = &LaserRENRhoEDotDot;
double LaserRENRhoNDotDot;
double *LaserRENRhoNDotDotPtr = &LaserRENRhoNDotDot;
double SeparationAngle;
double *SeparationAnglePtr = &SeparationAngle;
double SepAngleDot;
double *SepAngleDotPtr = &SepAngleDot;
double SepAngleDotDot;
double *SepAngleDotDotPtr = &SepAngleDotDot;


/**********************************************/
/*  GET SATELLITE EPHEMERIS INFORMATION       */
/**********************************************/
Sat->SetSSCNumber(SSCEdit->Text.ToInt());
strcpy(buff,ClassEdit->Text.c_str());
Sat->SetSecurityClass(buff);
strcpy(buff,IntIDEdit->Text.c_str());
Sat->SetInternationalID(buff);
Sat->SetEpochYear(EpochYearEdit->Text.ToInt());
Sat->SetEpochDay(EpochDayEdit->Text.ToDouble());
Sat->SetRevSquared(RevSquaredEdit->Text.ToDouble());
Sat->SetRevCubed(RevCubedEdit->Text.ToDouble());
Sat->SetBStarDrag(BStarEdit->Text.ToDouble());
Sat->SetEphemerisType(EphemerisTypeEdit->Text.ToInt());
Sat->SetElementSetNumber(ElSetEdit->Text.ToInt());
Sat->SetInclination(InclinationEdit->Text.ToDouble());
Sat->SetRightAscension(RightAscensionEdit->Text.ToDouble());
Sat->SetEccentricity(EccentricityEdit->Text.ToDouble());
Sat->SetArgumentOfPerigee(ArgumentOfPerigeeEdit->Text.ToDouble());
Sat->SetMeanAnomaly(MeanAnomalyEdit->Text.ToDouble());
Sat->SetMeanMotion(MeanMotionEdit->Text.ToDouble());
Sat->SetRevAtEpoch(RevNumberEdit->Text.ToInt());

/**********************************************/
/*  GET AIRCRAFT POSITION INFORMATION         */
/**********************************************/
ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
strcpy(buff,HemisphereEdit->Text.c_str());
if ((!(strcmp(buff, "N"))) || (!(strcmp(buff, "n"))))
    ABLPlatform->SetLatitudeHemisphere(0);
else if ((!(strcmp(buff, "S"))) || (!(strcmp(buff, "s"))))
    ABLPlatform->SetLatitudeHemisphere(1);
else
{    ErrorList.AddError("EvaluateEphemerisForm",
                        "Lat Hemisphere must be north(N) or south(S)",
                         1);
}
ABLPlatform->SetLatitudeDegree(LatitudeDegreeEdit->Text.ToDouble());
ABLPlatform->SetLatitudeMinute(LatitudeMinuteEdit->Text.ToDouble());
ABLPlatform->SetLatitudeSecond(LatitudeSecondEdit->Text.ToDouble());
```

```
    ABLPlatform->SetLongitudeDegree(LongitudeDegreeEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeMinute(LongitudeMinuteEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeSecond(LongitudeSecondEdit->Text.ToDouble());
    ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
    ABLPlatform->SetVelocityX(VelocityXEdit->Text.ToDouble());
    ABLPlatform->SetVelocityY(VelocityYEdit->Text.ToDouble());
    ABLPlatform->SetVelocityZ(VelocityZEdit->Text.ToDouble());

    /**********************************************/
    /*  GET GREENWICH MERIDIAN REFERENCE          */
    /**********************************************/
    ReferenceHour = ReferenceHourEdit->Text.ToDouble();
    ReferenceMinute = ReferenceMinuteEdit->Text.ToDouble();
    ReferenceSecond = ReferenceSecondEdit->Text.ToDouble();
    RefModJulianDate = RefModJulianDateEdit->Text.ToDouble();

    /**********************************************/
    /*   GET CURRENT TIME                         */
    /**********************************************/
    CalcYear = CalcYearEdit->Text.ToInt();
    CalcMonth = CalcMonthEdit->Text.ToInt();
    CalcDay = CalcDayEdit->Text.ToInt();
    CalcHour = CalcHourEdit->Text.ToInt();
    CalcMinute = CalcMinuteEdit->Text.ToInt();
    CalcSecond = CalcSecondEdit->Text.ToDouble();

    /**********************************************/
    /*   GET OTHER INPUTS                         */
    /**********************************************/
    SatPositionErrorInMeters = SatPosErrorEdit->Text.ToDouble();
    PlatformPositionErrorInMeters = PlatformPosErrorEdit->Text.ToDouble();
    MissilePositionErrorInMeters = MissilePosErrorEdit->Text.ToDouble();
    RangeToMissileInKilometers = MissileRangeEdit->Text.ToDouble();
    OtherErrorAngleInDeg = OtherErrorsEdit->Text.ToDouble();
    LaserAzimuthInDegrees = LaserAzimuthEdit->Text.ToDouble();
    LaserElevationInDegrees = LaserElevationEdit->Text.ToDouble();
    LaserAzimuthDot = LaserAzimuthDotEdit->Text.ToDouble();
    LaserElevationDot = LaserElevationDotEdit->Text.ToDouble();
    LaserAzimuthDotDot = LaserAzimuthDotDotEdit->Text.ToDouble();
    LaserElevationDotDot = LaserElevationDotDotEdit->Text.ToDouble();

/****************************************************/
/*  FIND THE CURRENT ANGLE OF THETA G AT THE        */
/*  TIME OF PROPAGATION                             */
/****************************************************/
    ThetaGInRadians = 0;
    FindThetaG(ReferenceHour,
               ReferenceMinute,
               ReferenceSecond,
               RefModJulianDate,
               CalcYear,
               CalcMonth,
               CalcDay,
               CalcHour,
               CalcMinute,
               CalcSecond,
               *ThetaPtr,
               *ErrorPtr);

/****************************************************/
/*  CONVERT THE PROPAGATION TIME TO A JULIAN DATE   */
/*  THAT CAN BE RECOGNIZED BY "TargetSatellite".    */
/****************************************************/
```

```
    JulianDate = 0.0;
    ConvertCalenderToJulian(CalcYear,
                            CalcMonth,
                            CalcDay,
                            CalcHour,
                            CalcMinute,
                            CalcSecond,
                            *JulianDatePtr,
                            *ErrorPtr);

/****************************************************/
/*  THIS IS THE MAIN MODULE BEING TESTED HERE.      */
/*  IT FINDS THE SEPARATION ANGLE AND THE RATE      */
/*  OF CHANGE AND ACCEL. OF THE ANGLE BETWEEN TWO   */
/*  VECTORS.                                        */
/****************************************************/
FindDisplacementAngles(*ABLPlatform,
                       *Sat,
                       *ThetaPtr,
                       JulianDate,
                       LaserAzimuthInDegrees,
                       LaserAzimuthDot,
                       LaserAzimuthDotDot,
                       LaserElevationInDegrees,
                       LaserElevationDot,
                       LaserElevationDotDot,
                       SatPositionErrorInMeters,
                       PlatformPositionErrorInMeters,
                       MissilePositionErrorInMeters,
                       RangeToMissileInKilometers,
                       OtherErrorAngleInDeg,
                       *PlatformSatRENRhoRPtr,
                       *PlatformSatRENRhoEPtr,
                       *PlatformSatRENRhoNPtr,
                       *PlatformSatRENRhoRDotPtr,
                       *PlatformSatRENRhoEDotPtr,
                       *PlatformSatRENRhoNDotPtr,
                       *PlatformSatRENRhoRDotDotPtr,
                       *PlatformSatRENRhoEDotDotPtr,
                       *PlatformSatRENRhoNDotDotPtr,
                       *LaserRENRhoRPtr,
                       *LaserRENRhoEPtr,
                       *LaserRENRhoNPtr,
                       *LaserRENRhoRDotPtr,
                       *LaserRENRhoEDotPtr,
                       *LaserRENRhoNDotPtr,
                       *LaserRENRhoRDotDotPtr,
                       *LaserRENRhoEDotDotPtr,
                       *LaserRENRhoNDotDotPtr,
                       *RangeToSatInKilometersPtr,
                       *ErrorAngleInRadiansPtr,
                       *SeparationAnglePtr,
                       *SepAngleDotPtr,
                       *SepAngleDotDotPtr,
                       *ErrorPtr);


/****************************************************/
/*  OUTPUT THE TEST PARAMETERS WHICH MONITOR THE    */
/*  CALCULATIONS IN "FindDisplacementAngle".        */
/****************************************************/
    RangeToSatEdit->Text = String(RangeToSatInKilometers);
    ErrorAngleEdit->Text = String(ErrorAngleInRadians * RADTODEGREES);
```

```
    SatREdit->Text = String(PlatformSatRENRhoR);
    SatEEdit->Text = String(PlatformSatRENRhoE);
    SatNEdit->Text = String(PlatformSatRENRhoN);
    SatRDotEdit->Text = String(PlatformSatRENRhoRDot);
    SatEDotEdit->Text = String(PlatformSatRENRhoEDot);
    SatNDotEdit->Text = String(PlatformSatRENRhoNDot);
    SatRDotDotEdit->Text        =        String(PlatformSatRENRhoRDotDot        *
1000.0);/*CONVERT*/
    SatEDotDotEdit->Text  =  String(PlatformSatRENRhoEDotDot  *  1000.0);/*KM  TO
M*/
    SatNDotDotEdit->Text   =   String(PlatformSatRENRhoNDotDot   *   1000.0);/*
*/
    LaserREdit->Text = String(LaserRENRhoR);
    LaserEEdit->Text = String(LaserRENRhoE);
    LaserNEdit->Text = String(LaserRENRhoN);
    LaserRDotEdit->Text = String(LaserRENRhoRDot);
    LaserEDotEdit->Text = String(LaserRENRhoEDot);
    LaserNDotEdit->Text = String(LaserRENRhoNDot);
    LaserRDotDotEdit->Text = String(LaserRENRhoRDotDot);
    LaserEDotDotEdit->Text = String(LaserRENRhoEDotDot);
    LaserNDotDotEdit->Text = String(LaserRENRhoNDotDot);
    SepAngleEdit->Text = String(SeparationAngle * RADTODEGREES);
    SepDotEdit->Text = String(SepAngleDot * RADTODEGREES);
    SepDotDotEdit->Text = String(SepAngleDotDot *RADTODEGREES);

/******************************************/
/*  PRINT OUT ALL ERROR MESSAGES          */
/******************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }


}

/************************************************/
/*  THIS EVENT HANDLER READS THE FIRST SATELLITE*/
/*  FROM A FILE OF TWO-LINE ELEMENT SETS        */
/************************************************/
void __fastcall TForm1::FileButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
    SatStructure *SatArray = new SatStructure;

    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int i;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    char FileName[MAXNAMELENGTH] = " ";


/****************************************************/
/*  GET NAME OF FILE TO READ TEST CASE FROM         */
/****************************************************/
```

```
    strcpy(FileName,FileEdit->Text.c_str());

/****************************************************/
/*  READ ALL SATELLITES FROM THE FILE, AND USE THE */
/*  FIRST SATELLITE IN THE FILE AS THE TEST CASE   */
/****************************************************/
    ReadTLEFile(FileName,
                *SatArray,
                *ErrorPtr);

/****************************************************/
/*  NOTE THE Sat[0] IS THE FIRST SATELLITE IN THE  */
/*  FILE                                           */
/****************************************************/
    SSCEdit->Text = String(SatArray->Sat[0].GetSSCNumber());
    ClassEdit->Text = String(SatArray->Sat[0].GetSecurityClass());
    IntIDEdit->Text = String(SatArray->Sat[0].GetInternationalID());
    EpochYearEdit->Text = String(SatArray->Sat[0].GetEpochYear());
    EpochDayEdit->Text = String(double(SatArray->Sat[0].GetEpochDay()));
    RevSquaredEdit->Text = String(double(SatArray->Sat[0].GetRevSquared()));
    RevCubedEdit->Text = String(double(SatArray->Sat[0].GetRevCubed()));
    BStarEdit->Text = String(double(SatArray->Sat[0].GetBStarDrag()));
    EphemerisTypeEdit->Text = String(SatArray->Sat[0].GetEphemerisType());
    ElSetEdit->Text = String(SatArray->Sat[0].GetElementSetNumber());
    InclinationEdit->Text = String(double(SatArray->Sat[0].GetInclination()));
    RightAscensionEdit->Text              =              String(double(SatArray-
>Sat[0].GetRightAscension()));
    EccentricityEdit->Text               =              String(double(SatArray-
>Sat[0].GetEccentricity()));
    ArgumentOfPerigeeEdit->Text            =            String(double(SatArray-
>Sat[0].GetArgumentOfPerigee()));
    MeanAnomalyEdit->Text = String(double(SatArray->Sat[0].GetMeanAnomaly()));
    MeanMotionEdit->Text = String(double(SatArray->Sat[0].GetMeanMotion()));
    RevNumberEdit->Text = String(SatArray->Sat[0].GetRevAtEpoch());


/****************************************************/
/*    DISPLAY ALL ERRORS                           */
/****************************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }

}
```

E.3 MainProcessorForm.cpp

```
/***************************************************************************/
/*  MODULE NAME:    MainProcessorForm.cpp                                  */
/*  AUTHOR:         Captain David Vloedman                                 */
/*  DATE CREATED:   January 10 , 1998                                      */
/*                                                                         */
/*  PURPOSE:        This is the Form which can be used to test the ABLPA   */
/*                  Main Processor.  This is the main Graphical User       */
/*                  Interface (GUI) for the Main Processor software.       */
/*                                                                         */
/*  COMPILER:       Borland C++ Builder3 Standard version                  */
/*                  This compiler should be used to compile and link.      */
/*                                                                         */
/***************************************************************************/
/*********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/*********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/*********************************/
/* USER-BUILT LIBRARIES          */
/*********************************/
#include "TimeModules.h"
#include "LaserConstants.h"
#include "ErrorStructure.h"
#include "TargetSatellite.h"
#include "TargetLaser.h"
#include "Aircraft.h"
#include "Satellite.h"
#include "EvaluateEphemerisModules.h"
#include "FindDisplacementAngleModules.h"
#include "MainProcessorForm.h"
#include "PAMainprocessor.h"
#include "ProcessSatellite.h"
#include "TLEInput.h"
/*********************************/
/* C SPECIFIC LIBRARIES          */
/*********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>

/*******************************/
/*      CREATE THE FORM        */
/*******************************/
TForm1 *Form1;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

/****************************************************/
/*  THIS EVENT-HANDLER EXECUTES THE MAIN PROCESSOR  */
/****************************************************/
void __fastcall TForm1::ProcessTLEFileButtonClick(TObject *Sender)
```

```
{
    ErrorStructure ErrorList;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    Aircraft* ABLPlatform;
        ABLPlatform = new Aircraft;
    SatStructure *SatArray = new SatStructure;

    char     Errors[MAXERRORS][MAXMESSAGELENGTH];
    char     buff[MAXNAMELENGTH];
    double   ReferenceHour;
    double   ReferenceMinute;
    double   ReferenceSecond;
    double   RefModJulianDate;
    int      CalcYear;
    int      CalcMonth;
    int      CalcDay;
    int      CalcHour;
    int      CalcMinute;
    double   CalcSecond;
    int      i;
    int      InFileLength;
    int      *InFileLengthPtr = &InFileLength;
    int      OutFileLength;
    int      *OutFileLengthPtr = &OutFileLength;
    int      ClosestApproachLength;
    int      *ClosestApproachLengthPtr = &ClosestApproachLength;
    char     InFileName[MAXNAMELENGTH] = " ";
    char     OutFileName[MAXNAMELENGTH] = " ";
    char     ClosestApproachFileName[MAXNAMELENGTH] = " ";
    char     buffer[MAXMESSAGELENGTH] = " ";
    double   ThetaGInDegrees;
    double   *ThetaPtr = &ThetaGInDegrees;
    double   LaserAzimuthInDegrees;
    double   LaserAzimuthDot;
    double   LaserAzimuthDotDot;
    double   LaserElevationInDegrees;
    double   LaserElevationDot;
    double   LaserElevationDotDot;
    double   SatPositionErrorInMeters;
    double   PlatformPositionErrorInMeters;
    double   MissilePositionErrorInMeters;
    double   RangeToMissileInKilometers;
    double   OtherErrorAngleInDeg;
    double   LazeDuration;
    double   SecondsFromVertex;
    double   InterpolationIncrement;

/*********************************************/
/*  GET THE NAMES OF THE INPUT AND OUTPUT    */
/*  FILES.                                   */
/*********************************************/
    strcpy(InFileName,InFileEdit->Text.c_str());
    strcpy(OutFileName,OutFileEdit->Text.c_str());
    strcpy(ClosestApproachFileName,CloseApproachFileEdit->Text.c_str());

/*********************************************/
/*  GET AIRCRAFT POSITION INFORMATION        */
/*********************************************/
    ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
    strcpy(buff,HemisphereEdit->Text.c_str());
    if ((!(strcmp(buff, "N"))) || (!(strcmp(buff, "n"))))
        ABLPlatform->SetLatitudeHemisphere(0);
    else if ((!(strcmp(buff, "S"))) || (!(strcmp(buff, "s"))))
```

```
        ABLPlatform->SetLatitudeHemisphere(1);
    else
    {    ErrorList.AddError("PAProcessorForm",
                           "Lat Hemisphere must be north(N) or south(S)",
                            1);
    }
    ABLPlatform->SetLatitudeDegree(LatitudeDegreeEdit->Text.ToDouble());
    ABLPlatform->SetLatitudeMinute(LatitudeMinuteEdit->Text.ToDouble());
    ABLPlatform->SetLatitudeSecond(LatitudeSecondEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeDegree(LongitudeDegreeEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeMinute(LongitudeMinuteEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeSecond(LongitudeSecondEdit->Text.ToDouble());
    ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
    ABLPlatform->SetVelocityX(VelocityXEdit->Text.ToDouble());
    ABLPlatform->SetVelocityY(VelocityYEdit->Text.ToDouble());
    ABLPlatform->SetVelocityZ(VelocityZEdit->Text.ToDouble());

/********************************************/
/*  GET GREENWICH MERIDIAN REFERENCE        */
/********************************************/
    ReferenceHour = ReferenceHourEdit->Text.ToDouble();
    ReferenceMinute = ReferenceMinuteEdit->Text.ToDouble();
    ReferenceSecond = ReferenceSecondEdit->Text.ToDouble();
    RefModJulianDate = RefModJulianDateEdit->Text.ToDouble();

/********************************************/
/*   GET CURRENT TIME                       */
/********************************************/
    CalcYear = CalcYearEdit->Text.ToInt();
    CalcMonth = CalcMonthEdit->Text.ToInt();
    CalcDay = CalcDayEdit->Text.ToInt();
    CalcHour = CalcHourEdit->Text.ToInt();
    CalcMinute = CalcMinuteEdit->Text.ToInt();
    CalcSecond = CalcSecondEdit->Text.ToDouble();
    LazeDuration = LazeDurationEdit->Text.ToDouble();

/************************************************/
/*   GET OTHER INPUTS INCLUDING LASER POSITION  */
/*   AND ERROR ANGLE INFORMATION                */
/************************************************/
    SatPositionErrorInMeters = SatPosErrorEdit->Text.ToDouble();
    PlatformPositionErrorInMeters = PlatformPosErrorEdit->Text.ToDouble();
    MissilePositionErrorInMeters = MissilePosErrorEdit->Text.ToDouble();
    RangeToMissileInKilometers = MissileRangeEdit->Text.ToDouble();
    OtherErrorAngleInDeg = OtherErrorsEdit->Text.ToDouble();
    LaserAzimuthInDegrees = LaserAzimuthEdit->Text.ToDouble();
    LaserElevationInDegrees = LaserElevationEdit->Text.ToDouble();
    LaserAzimuthDot = LaserAzimuthDotEdit->Text.ToDouble();
    LaserElevationDot = LaserElevationDotEdit->Text.ToDouble();
    LaserAzimuthDotDot = LaserAzimuthDotDotEdit->Text.ToDouble();
    LaserElevationDotDot = LaserElevationDotDotEdit->Text.ToDouble();
    SecondsFromVertex = VertexIntervalEdit->Text.ToDouble();
    InterpolationIncrement = InterpolationIncrementEdit->Text.ToDouble();

/********************************************/
/*  RUN THE PROCESSOR ON THE INPUT FILE     */
/********************************************/
    PAMainProcessor(InFileName,
                    OutFileName,
                    ClosestApproachFileName,
                    *InFileLengthPtr,
                    *OutFileLengthPtr,
                    *ClosestApproachLengthPtr,
```

```
                              *ABLPlatform,
                              ReferenceHour,
                              ReferenceMinute,
                              ReferenceSecond,
                              RefModJulianDate,
                              CalcYear,
                              CalcMonth,
                              CalcDay,
                              CalcHour,
                              CalcMinute,
                              CalcSecond,
                              LazeDuration,
                              LaserAzimuthInDegrees,
                              LaserAzimuthDot,
                              LaserAzimuthDotDot,
                              LaserElevationInDegrees,
                              LaserElevationDot,
                              LaserElevationDotDot,
                              SatPositionErrorInMeters,
                              PlatformPositionErrorInMeters,
                              MissilePositionErrorInMeters,
                              RangeToMissileInKilometers,
                              OtherErrorAngleInDeg,
                              SecondsFromVertex,
                              InterpolationIncrement,
                              *ThetaPtr,
                              *ErrorPtr);

/************************************************/
/*  DISPLAY THE NUMBER OF SATELLITE EPHEMERIDES */
/*  READ IN, AND HOW MANY SATELLITES WERE       */
/*  INTERSECTED.                                */
/************************************************/
    SatEvalEdit->Text = String(InFileLength);
    IntersectEdit->Text = String(OutFileLength);
    ThetaGEdit->Text = String(ThetaGInDegrees);

/**********************************************/
/*  THE "OutFile" CONTAINS ALL OF THE         */
/*  SATELLITE TLEs OF THE SATELLITES THAT ARE*/
/*  INTERSECTED.  NOW READ THE OUTFILE TO GET*/
/*  ALL THE SATELLITES INTERSECTED.           */
/**********************************************/
    ReadTLEFile(OutFileName,
                *SatArray,
                *ErrorPtr);

/**********************************************/
/*  SCROLL THROUGH ALL THE SATS INTERSECTED  */
/*  AND SHOW THEM ON THE SCREEN IN A MEMO BOX*/
/**********************************************/
    IntersectMemoBox->Lines->Clear();
    if (SatArray->NumSats == 0)
    {   sprintf(buffer,"No Satellites Intersected");
        IntersectMemoBox->Lines->Add(buffer);
    }
    else
    {   for (i=0; i<SatArray->NumSats; i++)
        {
            sprintf(buffer,"SSC: %d",
                    SatArray->Sat[i].GetSSCNumber());
            IntersectMemoBox->Lines->Add(buffer);
        }
```

```
    }

/*********************************************/
/*  THE "ClosestApproach" CONTAINS ALL OF THE*/
/*  SATELLITE TLEs OF THE SATELLITES THAT ARE*/
/*  CLOSE TO THE LASER.  NOW READ THE OUTFILE*/
/*  TO GET ALL THE CLOSE SATELLITES.         */
/*********************************************/
    ReadTLEFile(ClosestApproachFileName,
                *SatArray,
                *ErrorPtr);

/*********************************************/
/*  SCROLL THROUGH ALL THE CLOSE SATS AND    */
/*  SHOW THEM ON THE SCREEN IN A MEMO BOX    */
/*********************************************/
    InterpolateMemoBox->Lines->Clear();
    if (SatArray->NumSats == 0)
    {   sprintf(buffer,"No Satellites Interpolated");
        InterpolateMemoBox->Lines->Add(buffer);
    }
    else
    {   for (i=0; i<SatArray->NumSats; i++)
        {
            sprintf(buffer,"SSC: %d",
                    SatArray->Sat[i].GetSSCNumber());
            InterpolateMemoBox->Lines->Add(buffer);
        }
    }

/*********************************************/
/*  DISPLAY ANY ERRORS THAT HAVE OCCURRED    */
/*********************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }


}
```

E.4 PAPreprocessorForm.cpp

```
/******************************************************************************/
/*  MODULE NAME:     PAPreprocessorForm.cpp                                   */
/*  AUTHOR:          Captain David Vloedman                                   */
/*  DATE CREATED:    October 12, 1998                                         */
/*                                                                            */
/*  PURPOSE:         This is the Form which can be used as the Front End to  */
/*                   the PAPreprocessor module.                               */
/*                                                                            */
/*  COMPILER:        Borland C++ Builder3 Standard version                    */
/*                   This compiler should be used to compile and link.        */
/*                                                                            */
/******************************************************************************/
/*********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/*********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/*********************************/
/* USER-BUILT LIBRARIES          */
/*********************************/
#include "LaserConstants.h"
#include "ErrorStructure.h"
#include "PAPreprocessorForm.h"
#include "Aircraft.h"
#include "Satellite.h"
#include "PAPreprocessor.h"
#include "TLEInput.h"
/*********************************/
/* C SPECIFIC LIBRARIES          */
/*********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>

/*****************************/
/*      CREATE THE FORM      */
/*****************************/
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

/*************************************************************/
/*  THIS EVENT HANDLER BASICALLY HANDLES THE RUNNING OF THE    */
/*  PREPROCESSOR.  IT IS ACTIVATED BY PRESSING THE "Evaluate   */
/*  TLE File" BUTTON.                                          */
/*************************************************************/
void __fastcall TForm1::EvaluateTLEFileButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    Aircraft* ABLPlatform;
        ABLPlatform = new Aircraft;
    SatStructure *SatArray = new SatStructure;
```

```
    char     Errors[MAXERRORS][MAXMESSAGELENGTH];
    char     buff[MAXNAMELENGTH];
    double   ReferenceHour;
    double   ReferenceMinute;
    double   ReferenceSecond;
    double   RefModJulianDate;
    int      CalcYear;
    int      CalcMonth;
    int      CalcDay;
    int      CalcHour;
    int      CalcMinute;
    double   CalcSecond;
    int      i;
    int      InFileLength;
    int      *InFileLengthPtr = &InFileLength;
    int      OutFileLength;
    int      *OutFileLengthPtr = &OutFileLength;
    char     InFileName[MAXNAMELENGTH] = " ";
    char     OutFileName[MAXNAMELENGTH] = " ";
    char     buffer[MAXMESSAGELENGTH] = " ";
    double   TimeToNextRun;
    double   ThetaGInDegrees;
    double   *ThetaPtr = &ThetaGInDegrees;


/*********************************************/
/*  GET THE NAMES OF THE INPUT AND OUTPUT    */
/*  FILES.                                   */
/*********************************************/
    strcpy(InFileName,InFileEdit->Text.c_str());
    strcpy(OutFileName,OutFileEdit->Text.c_str());

/*********************************************/
/*  GET AIRCRAFT POSITION INFORMATION        */
/*********************************************/
    ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
    strcpy(buff,HemisphereEdit->Text.c_str());
    if ((!(strcmp(buff, "N"))) || (!(strcmp(buff, "n"))))
        ABLPlatform->SetLatitudeHemisphere(0);
    else if ((!(strcmp(buff, "S"))) || (!(strcmp(buff, "s"))))
        ABLPlatform->SetLatitudeHemisphere(1);
    else
    {    ErrorList.AddError("PAProcessorForm",
                             "Lat Hemisphere must be north(N) or south(S)",
                              1);
    }
    ABLPlatform->SetLatitudeDegree(LatitudeDegreeEdit->Text.ToDouble());
    ABLPlatform->SetLatitudeMinute(LatitudeMinuteEdit->Text.ToDouble());
    ABLPlatform->SetLatitudeSecond(LatitudeSecondEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeDegree(LongitudeDegreeEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeMinute(LongitudeMinuteEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeSecond(LongitudeSecondEdit->Text.ToDouble());
    ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
    ABLPlatform->SetVelocityX(VelocityXEdit->Text.ToDouble());
    ABLPlatform->SetVelocityY(VelocityYEdit->Text.ToDouble());
    ABLPlatform->SetVelocityZ(VelocityZEdit->Text.ToDouble());

/*********************************************/
/*  GET GREENWICH MERIDIAN REFERENCE         */
/*********************************************/
    ReferenceHour = ReferenceHourEdit->Text.ToDouble();
    ReferenceMinute = ReferenceMinuteEdit->Text.ToDouble();
```

```
    ReferenceSecond = ReferenceSecondEdit->Text.ToDouble();
    RefModJulianDate = RefModJulianDateEdit->Text.ToDouble();
    TimeToNextRun = TimeToRunEdit->Text.ToDouble();

/***********************************************/
/*   GET CURRENT TIME                          */
/***********************************************/
    CalcYear = CalcYearEdit->Text.ToInt();
    CalcMonth = CalcMonthEdit->Text.ToInt();
    CalcDay = CalcDayEdit->Text.ToInt();
    CalcHour = CalcHourEdit->Text.ToInt();
    CalcMinute = CalcMinuteEdit->Text.ToInt();
    CalcSecond = CalcSecondEdit->Text.ToDouble();

/***********************************************/
/*  RUN THE PREPROCESSOR ON THE INPUT FILE    */
/***********************************************/
    PAPreprocessor( InFileName,
                    OutFileName,
                    *InFileLengthPtr,
                    *OutFileLengthPtr,
                    *ABLPlatform,
                    ReferenceHour,
                    ReferenceMinute,
                    ReferenceSecond,
                    RefModJulianDate,
                    CalcYear,
                    CalcMonth,
                    CalcDay,
                    CalcHour,
                    CalcMinute,
                    CalcSecond,
                    TimeToNextRun,
                    *ThetaPtr,
                    *ErrorPtr);

/**************************************************/
/*  DISPLAY THE NUMBER OF SATELLITE EPHEMERIDES */
/*  READ IN, AND HOW MANY WERE "IN VIEW".        */
/**************************************************/
    SatEvalEdit->Text = String(InFileLength);
    InRangeEdit->Text = String(OutFileLength);
    ThetaGEdit->Text = String(ThetaGInDegrees);

/***********************************************/
/*  THE "OutFile" CONTAINS ALL OF THE          */
/*  SATELLITE TLEs OF THE SATELLITES THAT ARE*/
/*  IN VIEW.  NOW READ THE OUTFILE TO GET      */
/*  ALL THE SATELLITES IN VIEW.                */
/***********************************************/
    ReadTLEFile(OutFileName,
                *SatArray,
                *ErrorPtr);

/***********************************************/
/*  SCROLL THROUGH ALL THE SATS IN VIEW AND   */
/*  SHOW THEM ON THE SCREEN IN A MEMO BOX     */
/***********************************************/
    InRangeMemoBox->Lines->Clear();
    if (SatArray->NumSats == 0)
    {   sprintf(buffer,"No Satellites In Range");
        InRangeMemoBox->Lines->Add(buffer);
    }
```

```
    else
    {   for (i=0; i<SatArray->NumSats; i++)
        {
            sprintf(buffer,"SSC: %d",
                    SatArray->Sat[i].GetSSCNumber());
            InRangeMemoBox->Lines->Add(buffer);
        }
    }

/*********************************************/
/*  DISPLAY ANY ERRORS THAT HAVE OCCURRED    */
/*********************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }


}
```

E.5 ProcessSatelliteForm.cpp

```
/****************************************************************************/
/*  MODULE NAME:     ProcessSatelliteForm.cpp                               */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    January 10 , 1998                                      */
/*                                                                          */
/*  PURPOSE:         This is the Form which can be used to test the modules */
/*                   created in ProcessSatellite.cpp.  This form            */
/*                   takes all the inputs to evaluate a single satellite    */
/*                   ephemeris against a single airborne platform, and      */
/*                   determines the separation angle of the satellite pos   */
/*                   vector with respect to the ABL laser beamas well as the */
/*                   time to intersect.                                     */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "TimeModules.h"
#include "LaserConstants.h"
#include "ErrorStructure.h"
#include "TargetSatellite.h"
#include "TargetLaser.h"
#include "Aircraft.h"
#include "Satellite.h"
#include "EvaluateEphemerisModules.h"
#include "FindDisplacementAngleModules.h"
#include "ProcessSatelliteForm.h"
#include "ProcessSatellite.h"
#include "TLEInput.h"
/**********************************/
/* C SPECIFIC LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
/********************************/
/*      CREATE THE FORM        */
/********************************/
TForm1 *Form1;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
/***************************************************/
/*  THIS EVENT HANDLER READS SATELLITE INFORMATION */
/*  FROM A FILE                                    */
```

```
/*************************************************/
void __fastcall TForm1::FileButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
    SatStructure *SatArray = new SatStructure;

    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int i;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    char FileName[MAXNAMELENGTH] = " ";


/*************************************************/
/*  GET NAME OF FILE TO READ TEST CASE FROM      */
/*************************************************/
    strcpy(FileName,FileEdit->Text.c_str());

/*************************************************/
/*  READ ALL SATELLITES FROM THE FILE, AND USE THE */
/*  FIRST SATELLITE IN THE FILE AS THE TEST CASE   */
/*************************************************/
    ReadTLEFile(FileName,
                *SatArray,
                *ErrorPtr);

/*************************************************/
/*  NOTE THE Sat[0] IS THE FIRST SATELLITE IN THE  */
/*  FILE                                           */
/*************************************************/
    SSCEdit->Text = String(SatArray->Sat[0].GetSSCNumber());
    ClassEdit->Text = String(SatArray->Sat[0].GetSecurityClass());
    IntIDEdit->Text = String(SatArray->Sat[0].GetInternationalID());
    EpochYearEdit->Text = String(SatArray->Sat[0].GetEpochYear());
    EpochDayEdit->Text = String(double(SatArray->Sat[0].GetEpochDay()));
    RevSquaredEdit->Text = String(double(SatArray->Sat[0].GetRevSquared()));
    RevCubedEdit->Text = String(double(SatArray->Sat[0].GetRevCubed()));
    BStarEdit->Text = String(double(SatArray->Sat[0].GetBStarDrag()));
    EphemerisTypeEdit->Text = String(SatArray->Sat[0].GetEphemerisType());
    ElSetEdit->Text = String(SatArray->Sat[0].GetElementSetNumber());
    InclinationEdit->Text = String(double(SatArray->Sat[0].GetInclination()));
    RightAscensionEdit->Text              =              String(double(SatArray-
>Sat[0].GetRightAscension()));
    EccentricityEdit->Text               =               String(double(SatArray-
>Sat[0].GetEccentricity()));
    ArgumentOfPerigeeEdit->Text            =             String(double(SatArray-
>Sat[0].GetArgumentOfPerigee()));
    MeanAnomalyEdit->Text = String(double(SatArray->Sat[0].GetMeanAnomaly()));
    MeanMotionEdit->Text = String(double(SatArray->Sat[0].GetMeanMotion()));
    RevNumberEdit->Text = String(SatArray->Sat[0].GetRevAtEpoch());


/*************************************************/
/*    DISPLAY ALL ERRORS                         */
/*************************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
```

```
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }

}

/****************************************************/
/*  THIS EVENT HANDLER READS INPUT PARAMETERS      */
/*  AND CALLS THE MAIN "ProcessSatellite" ROUTINE  */
/****************************************************/
void __fastcall TForm1::EvaluateButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    Aircraft* ABLPlatform;
        ABLPlatform = new Aircraft;
    Satellite* Sat;
        Sat = new Satellite;
    char    Errors[MAXERRORS][MAXMESSAGELENGTH];
    char    buff[MAXNAMELENGTH];
    double  ReferenceHour;              /***************************/
    double  ReferenceMinute;            /*  THE REFERENT ANGLE     */
    double  ReferenceSecond;            /*  OF THETA G IS KNOWN AS */
    double  RefModJulianDate;           /*  THE REFERENCE ANGLE IN */
    double  ThetaGInRadians;            /***************************/
    double  *ThetaPtr = &ThetaGInRadians;
    double  ErrorAngleInRadians;
    double  *ErrorAngleInRadiansPtr = &ErrorAngleInRadians;
    double  LaserAzimuthInDegrees;
    double  LaserAzimuthDot;
    double  LaserAzimuthDotDot;
    double  LaserElevationInDegrees;
    double  LaserElevationDot;
    double  LaserElevationDotDot;
    double  SatPositionErrorInMeters;
    double  PlatformPositionErrorInMeters;
    double  MissilePositionErrorInMeters;
    double  RangeToMissileInKilometers;
    double  OtherErrorAngleInDeg;
    int     CalcYear;
    int     CalcMonth;
    int     CalcDay;
    int     CalcHour;
    int     CalcMinute;
    double  CalcSecond;
    double  LazeDuration;
    int     i;
    double JulianDate;
    double *JulianDatePtr = &JulianDate;
    double RangeToSatInKilometers;
    double *RangeToSatInKilometersPtr = &RangeToSatInKilometers;
    double SeparationAngle;
    double *SeparationAnglePtr = &SeparationAngle;
    double SepAngleDot;
    double *SepAngleDotPtr = &SepAngleDot;
    double SepAngleDotDot;
    double *SepAngleDotDotPtr = &SepAngleDotDot;
    int    Intersection;
    int    *IntersectionPtr = &Intersection;
    int    Interpolation;
    int    *InterpolationPtr = &Interpolation;
    double ClosestApproachInDegrees;
```

```
double *ClosestApproachInDegreesPtr = &ClosestApproachInDegrees;
double TimeToIntersect;
double *TimeToIntersectPtr = &TimeToIntersect;
double SecondsFromVertex;
double InterpolationIncrement;

/**********************************************/
/*  GET SATELLITE EPHEMERIS INFORMATION       */
/**********************************************/
Sat->SetSSCNumber(SSCEdit->Text.ToInt());
strcpy(buff,ClassEdit->Text.c_str());
Sat->SetSecurityClass(buff);
strcpy(buff,IntIDEdit->Text.c_str());
Sat->SetInternationalID(buff);
Sat->SetEpochYear(EpochYearEdit->Text.ToInt());
Sat->SetEpochDay(EpochDayEdit->Text.ToDouble());
Sat->SetRevSquared(RevSquaredEdit->Text.ToDouble());
Sat->SetRevCubed(RevCubedEdit->Text.ToDouble());
Sat->SetBStarDrag(BStarEdit->Text.ToDouble());
Sat->SetEphemerisType(EphemerisTypeEdit->Text.ToInt());
Sat->SetElementSetNumber(ElSetEdit->Text.ToInt());
Sat->SetInclination(InclinationEdit->Text.ToDouble());
Sat->SetRightAscension(RightAscensionEdit->Text.ToDouble());
Sat->SetEccentricity(EccentricityEdit->Text.ToDouble());
Sat->SetArgumentOfPerigee(ArgumentOfPerigeeEdit->Text.ToDouble());
Sat->SetMeanAnomaly(MeanAnomalyEdit->Text.ToDouble());
Sat->SetMeanMotion(MeanMotionEdit->Text.ToDouble());
Sat->SetRevAtEpoch(RevNumberEdit->Text.ToInt());

/**********************************************/
/*  GET AIRCRAFT POSITION INFORMATION         */
/**********************************************/
ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
strcpy(buff,HemisphereEdit->Text.c_str());
if ((!(strcmp(buff, "N"))) || (!(strcmp(buff, "n"))))
    ABLPlatform->SetLatitudeHemisphere(0);
else if ((!(strcmp(buff, "S"))) || (!(strcmp(buff, "s"))))
    ABLPlatform->SetLatitudeHemisphere(1);
else
{    ErrorList.AddError("EvaluateEphemerisForm",
                        "Lat Hemisphere must be north(N) or south(S)",
                         1);
}
ABLPlatform->SetLatitudeDegree(LatitudeDegreeEdit->Text.ToDouble());
ABLPlatform->SetLatitudeMinute(LatitudeMinuteEdit->Text.ToDouble());
ABLPlatform->SetLatitudeSecond(LatitudeSecondEdit->Text.ToDouble());
ABLPlatform->SetLongitudeDegree(LongitudeDegreeEdit->Text.ToDouble());
ABLPlatform->SetLongitudeMinute(LongitudeMinuteEdit->Text.ToDouble());
ABLPlatform->SetLongitudeSecond(LongitudeSecondEdit->Text.ToDouble());
ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
ABLPlatform->SetVelocityX(VelocityXEdit->Text.ToDouble());
ABLPlatform->SetVelocityY(VelocityYEdit->Text.ToDouble());
ABLPlatform->SetVelocityZ(VelocityZEdit->Text.ToDouble());

/**********************************************/
/*  GET GREENWICH MERIDIAN REFERENCE          */
/**********************************************/
ReferenceHour = ReferenceHourEdit->Text.ToDouble();
ReferenceMinute = ReferenceMinuteEdit->Text.ToDouble();
ReferenceSecond = ReferenceSecondEdit->Text.ToDouble();
RefModJulianDate = RefModJulianDateEdit->Text.ToDouble();

/**********************************************/
```

```
    /*   GET CURRENT TIME                              */
    /***********************************************/
    CalcYear = CalcYearEdit->Text.ToInt();
    CalcMonth = CalcMonthEdit->Text.ToInt();
    CalcDay = CalcDayEdit->Text.ToInt();
    CalcHour = CalcHourEdit->Text.ToInt();
    CalcMinute = CalcMinuteEdit->Text.ToInt();
    CalcSecond = CalcSecondEdit->Text.ToDouble();
    LazeDuration = LazeDurationEdit->Text.ToDouble();

    /***********************************************/
    /*   GET OTHER INPUTS                              */
    /***********************************************/
    SatPositionErrorInMeters = SatPosErrorEdit->Text.ToDouble();
    PlatformPositionErrorInMeters = PlatformPosErrorEdit->Text.ToDouble();
    MissilePositionErrorInMeters = MissilePosErrorEdit->Text.ToDouble();
    RangeToMissileInKilometers = MissileRangeEdit->Text.ToDouble();
    OtherErrorAngleInDeg = OtherErrorsEdit->Text.ToDouble();
    LaserAzimuthInDegrees = LaserAzimuthEdit->Text.ToDouble();
    LaserElevationInDegrees = LaserElevationEdit->Text.ToDouble();
    LaserAzimuthDot = LaserAzimuthDotEdit->Text.ToDouble();
    LaserElevationDot = LaserElevationDotEdit->Text.ToDouble();
    LaserAzimuthDotDot = LaserAzimuthDotDotEdit->Text.ToDouble();
    LaserElevationDotDot = LaserElevationDotDotEdit->Text.ToDouble();
    SecondsFromVertex = VertexIntervalEdit->Text.ToDouble();
    InterpolationIncrement = InterpolationIncrementEdit->Text.ToDouble();

/****************************************************/
/*  FIND THE CURRENT ANGLE OF THETA G AT THE        */
/*  TIME OF PROPAGATION                             */
/****************************************************/
    ThetaGInRadians = 0;
    FindThetaG(ReferenceHour,
               ReferenceMinute,
               ReferenceSecond,
               RefModJulianDate,
               CalcYear,
               CalcMonth,
               CalcDay,
               CalcHour,
               CalcMinute,
               CalcSecond,
               *ThetaPtr,
               *ErrorPtr);

/****************************************************/
/*  CONVERT THE PROPAGATION TIME TO A JULIAN DATE   */
/*  THAT CAN BE RECOGNIZED BY "TargetSatellite".    */
/****************************************************/
    JulianDate = 0.0;
    ConvertCalenderToJulian(CalcYear,
                            CalcMonth,
                            CalcDay,
                            CalcHour,
                            CalcMinute,
                            CalcSecond,
                            *JulianDatePtr,
                            *ErrorPtr);

/****************************************************/
/*  CALL "ProcessSatellite" MODULE TO FIND THE      */
/*  INTERSECTION ANGLES AND TIME                    */
/****************************************************/
```

```
    ProcessSatellite(*ABLPlatform,
                     *Sat,
                     ReferenceHour,
                     ReferenceMinute,
                     ReferenceSecond,
                     RefModJulianDate,
                     SecondsFromVertex,
                     InterpolationIncrement,
                     *ThetaPtr,
                     JulianDate,
                     LazeDuration,
                     LaserAzimuthInDegrees,
                     LaserAzimuthDot,
                     LaserAzimuthDotDot,
                     LaserElevationInDegrees,
                     LaserElevationDot,
                     LaserElevationDotDot,
                     SatPositionErrorInMeters,
                     PlatformPositionErrorInMeters,
                     MissilePositionErrorInMeters,
                     RangeToMissileInKilometers,
                     OtherErrorAngleInDeg,
                     *RangeToSatInKilometersPtr,
                     *ErrorAngleInRadiansPtr,
                     *SeparationAnglePtr,
                     *SepAngleDotPtr,
                     *SepAngleDotDotPtr,
                     *IntersectionPtr,
                     *InterpolationPtr,
                     *TimeToIntersectPtr,
                     *ClosestApproachInDegreesPtr,
                     *ErrorPtr);

/****************************************************/
/*  OUTPUT THE TEST PARAMETERS WHICH MONITOR THE    */
/*  CALCULATIONS IN "FindDisplacementAngle".        */
/****************************************************/
    RangeToSatEdit->Text = String(RangeToSatInKilometers);
    ErrorAngleEdit->Text = String(ErrorAngleInRadians * RADTODEGREES);
    SepAngleEdit->Text = String(SeparationAngle * RADTODEGREES);
    SepDotEdit->Text = String(SepAngleDot * RADTODEGREES);
    SepDotDotEdit->Text = String(SepAngleDotDot *RADTODEGREES);
    TimeToIntersectEdit->Text = String(TimeToIntersect);
    ClosestApproachEdit->Text = String(ClosestApproachInDegrees);

    if (Intersection == 1)
       IntersectionEdit->Text = "YES";
    else
       IntersectionEdit->Text = "NO";

    if (Interpolation == 1)
       InterpolationEdit->Text = "YES";
    else
       InterpolationEdit->Text = "NO";

/******************************************/
/*  PRINT OUT ALL ERROR MESSAGES          */
/******************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
```

```
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }


}
```

E.6 SGP4TestForm.cpp

```
/*****************************************************************************/
/*  MODULE NAME:      SGP4TestForm.cpp                                      */
/*  AUTHOR:           Captain David Vloedman                                */
/*  DATE CREATED:     October 10, 1998                                      */
/*                                                                          */
/*  PURPOSE:          This test form module is a test module for the routines */
/*                    handle calling of the satellite propagator. "SGP4". This*/
/*                    propagator is US Space Command's satellite time/position*/
/*                    propagator using general perturbations only.  The     */
/*                    version of SGP4 used here is version 3.01 in C        */
/*                                                                          */
/*                                                                          */
/*  COMPILER:         Borland C++ Builder3 Standard version                 */
/*                    This compiler should be used to compile and link.     */
/*                                                                          */
/*****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "SGP4TestForm.h"
#include "SGP4SupportModules.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "ErrorStructure.h"
#include "TLEInput.h"
/**********************************/
/* C STANDARD LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include "SGP4Routines.h"
#include "TimeModules.h"
/********************************/
/*      CREATE THE FORM        */
/********************************/
TForm1 *Form1;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
/****************************************************/
/*  THIS EVENT HANDLER PROCEDURE HANDLES THE BUTTON*/
/*  THAT CAN LOAD A TEST CASE FROM A FILE FOR LATER*/
/*  EXECUTION                                       */
/****************************************************/
void __fastcall TForm1::FileButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
```

```
    SatStructure *SatArray = new SatStructure;

    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int i;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    char FileName[MAXNAMELENGTH] = " ";


/****************************************************/
/*  GET NAME OF FILE TO READ TEST CASE FROM        */
/****************************************************/
    strcpy(FileName,FileEdit->Text.c_str());

/****************************************************/
/*  READ ALL SATELLITES FROM THE FILE, AND USE THE */
/*  FIRST SATELLITE IN THE FILE AS THE TEST CASE   */
/****************************************************/
    ReadTLEFile(FileName,
                *SatArray,
                *ErrorPtr);
/****************************************************/
/*  NOTE THE Sat[0] IS THE FIRST SATELLITE IN THE  */
/*  FILE                                           */
/****************************************************/
    SSCEdit->Text = String(SatArray->Sat[0].GetSSCNumber());
    ClassEdit->Text = String(SatArray->Sat[0].GetSecurityClass());
    IntIDEdit->Text = String(SatArray->Sat[0].GetInternationalID());
    EpochYearEdit->Text = String(SatArray->Sat[0].GetEpochYear());
    EpochDayEdit->Text = String(double(SatArray->Sat[0].GetEpochDay()));
    RevSquaredEdit->Text = String(double(SatArray->Sat[0].GetRevSquared()));
    RevCubedEdit->Text = String(double(SatArray->Sat[0].GetRevCubed()));
    BStarEdit->Text = String(double(SatArray->Sat[0].GetBStarDrag()));
    EphemerisTypeEdit->Text = String(SatArray->Sat[0].GetEphemerisType());
    ElSetEdit->Text = String(SatArray->Sat[0].GetElementSetNumber());
    InclinationEdit->Text = String(double(SatArray->Sat[0].GetInclination()));
    RightAscensionEdit->Text            =            String(double(SatArray-
>Sat[0].GetRightAscension()));
    EccentricityEdit->Text             =             String(double(SatArray-
>Sat[0].GetEccentricity()));
    ArgumentOfPerigeeEdit->Text            =            String(double(SatArray-
>Sat[0].GetArgumentOfPerigee()));
    MeanAnomalyEdit->Text = String(double(SatArray->Sat[0].GetMeanAnomaly()));
    MeanMotionEdit->Text = String(double(SatArray->Sat[0].GetMeanMotion()));
    RevNumberEdit->Text = String(SatArray->Sat[0].GetRevAtEpoch());


/****************************************************/
/*    DISPLAY ALL ERRORS                           */
/****************************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }

}
```

```
/**************************************************/
/*  THIS PROCEDURE ACTUALLY RUNS THE TEST CASE AS  */
/*  IT HAS BEEN ENTERED INTO THE FORM AND DISPLAYS */
/*  THE RESULTS OF THE RUN                         */
/**************************************************/
void __fastcall TForm1::RunButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    Satellite* Sat;
        Sat = new Satellite;
    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int i;
    char    buff[MAXNAMELENGTH];
    double JulianDate;
    double  Inclination;
    double  *InclinationPtr = &Inclination;
    double  RightAscension;
    double  *RightAscensionPtr = &RightAscension;
    double  Eccentricity;
    double  *EccentricityPtr = &Eccentricity;
    double  MeanMotion;
    double  *MeanMotionPtr = &MeanMotion;
    double  ArgumentOfPerigee;
    double  *ArgumentOfPerigeePtr = &ArgumentOfPerigee;
    double  MeanAnomaly;
    double  *MeanAnomalyPtr = &MeanAnomaly;
    double X;
    double *XPtr = &X;
    double Y;
    double *YPtr = &Y;
    double Z;
    double *ZPtr = &Z;
    double Xdot;
    double *XdotPtr = &Xdot;
    double Ydot;
    double *YdotPtr = &Ydot;
    double Zdot;
    double *ZdotPtr = &Zdot;
    double Delta;
    double *DeltaPtr = &Delta;


    /**********************************************/
    /*  GET SATELLITE EPHEMERIS INFORMATION       */
    /**********************************************/
    Sat->SetSSCNumber(SSCEdit->Text.ToInt());
    strcpy(buff,ClassEdit->Text.c_str());
    Sat->SetSecurityClass(buff);
    strcpy(buff,IntIDEdit->Text.c_str());
    Sat->SetInternationalID(buff);
    Sat->SetEpochYear(EpochYearEdit->Text.ToInt());
    Sat->SetEpochDay(EpochDayEdit->Text.ToDouble());
    Sat->SetRevSquared(RevSquaredEdit->Text.ToDouble());
    Sat->SetRevCubed(RevCubedEdit->Text.ToDouble());
    Sat->SetBStarDrag(BStarEdit->Text.ToDouble());
    Sat->SetEphemerisType(EphemerisTypeEdit->Text.ToInt());
    Sat->SetElementSetNumber(ElSetEdit->Text.ToInt());
    Sat->SetInclination(InclinationEdit->Text.ToDouble());
    Sat->SetRightAscension(RightAscensionEdit->Text.ToDouble());
    Sat->SetEccentricity(EccentricityEdit->Text.ToDouble());
```

```
    Sat->SetArgumentOfPerigee(ArgumentOfPerigeeEdit->Text.ToDouble());
    Sat->SetMeanAnomaly(MeanAnomalyEdit->Text.ToDouble());
    Sat->SetMeanMotion(MeanMotionEdit->Text.ToDouble());
    Sat->SetRevAtEpoch(RevNumberEdit->Text.ToInt());
    JulianDate = JulianDateEdit->Text.ToDouble();

/****************************************************/
/*  MAKE A CALL TO THE SGP4 PROPAGATOR              */
/****************************************************/
    CallSGP4(*Sat,
             JulianDate,
             *XPtr,
             *YPtr,
             *ZPtr,
             *XdotPtr,
             *YdotPtr,
             *ZdotPtr,
             *InclinationPtr,
             *RightAscensionPtr,
             *EccentricityPtr,
             *MeanMotionPtr,
             *ArgumentOfPerigeePtr,
             *MeanAnomalyPtr,
             *DeltaPtr,
             *ErrorPtr);

/**********************************************/
/*  Convert Mean Motion from radians/sec to   */
/*  revolutions per day                       */
/**********************************************/
    MeanMotion = MeanMotion * MINUTESPERDAY / TWOPI;

/****************************************************/
/*  DISPLAY THE RESULTS OBTAINED FROM SGP4          */
/****************************************************/
    XEdit->Text = String(X);
    YEdit->Text = String(Y);
    ZEdit->Text = String(Z);
    XdotEdit->Text = String(Xdot);
    YdotEdit->Text = String(Ydot);
    ZdotEdit->Text = String(Zdot);
    DeltaEdit->Text = String(Delta);
    InclinOutEdit->Text = String(Inclination);
    RightAsOutEdit->Text = String(RightAscension);
    EccentricityOutEdit->Text = String(Eccentricity);
    MeanMotionOutEdit->Text = String(MeanMotion);
    ArgOfPerigeeOutEdit->Text = String(ArgumentOfPerigee);
    MeanAnomalyOutEdit->Text = String(MeanAnomaly);
    DeltaEdit->Text = String(Delta);


/****************************************************/
/*    DISPLAY ALL ERRORS                            */
/****************************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
```

```
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }

}
```

E.7 TargetPlatformForm.cpp

```
/****************************************************************************/
/*  MODULE NAME:     TargetPlatformForm.cpp                                 */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    January 13, 1998                                       */
/*                                                                          */
/*  PURPOSE:         This is the Form which can be used to test the modules */
/*                   created in TargetPlatform.cpp.  This form              */
/*                   takes all the inputs to evaluate the position and      */
/*                   velocity of the aircraft in ECI and REN coordinates.   */
/*                   The conversion matrix is also returned, but not        */
/*                   displayed on the form.                                 */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/****************************************************************************/
/*********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/*********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/*********************************/
/* USER-BUILT LIBRARIES          */
/*********************************/
#include "TimeModules.h"
#include "LaserConstants.h"
#include "ErrorStructure.h"
#include "TargetPlatformForm.h"
#include "Aircraft.h"
#include "EvaluateEphemerisModules.h"
#include "TargetPlatform.h"
#include "TLEInput.h"
/*********************************/
/* C SPECIFIC LIBRARIES          */
/*********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>

/******************************/
/*      CREATE THE FORM       */
/******************************/
TForm1 *Form1;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
/****************************************************/
/*  THIS ERROR HANDLER CALLS THE "TargetPlatform"   */
/*  MODULE.                                         */
/****************************************************/
void __fastcall TForm1::EvaluateButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
```

```
ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
Aircraft* ABLPlatform;
    ABLPlatform = new Aircraft;
char     Errors[MAXERRORS][MAXMESSAGELENGTH];
char     buff[MAXNAMELENGTH];
double   ReferenceHour;              /****************************/
double   ReferenceMinute;            /*  THE REFERENT ANGLE      */
double   ReferenceSecond;            /*  OF THETA G IS KNOWN AS  */
double   RefModJulianDate;           /*  THE REFERENCE ANGLE IN  */
double   ThetaGInRadians;            /****************************/
double   *ThetaPtr = &ThetaGInRadians;
int      CalcYear;
int      CalcMonth;
int      CalcDay;
int      CalcHour;
int      CalcMinute;
double   CalcSecond;
int      i;
double JulianDate;
double *JulianDatePtr = &JulianDate;
double PlatformECIRhoX;
double *PlatformECIRhoXPtr = &PlatformECIRhoX;
double PlatformECIRhoY;
double *PlatformECIRhoYPtr = &PlatformECIRhoY;
double PlatformECIRhoZ;
double *PlatformECIRhoZPtr = &PlatformECIRhoZ;
double PlatformECIRhoXDot;
double *PlatformECIRhoXDotPtr = &PlatformECIRhoXDot;
double PlatformECIRhoYDot;
double *PlatformECIRhoYDotPtr = &PlatformECIRhoYDot;
double PlatformECIRhoZDot;
double *PlatformECIRhoZDotPtr = &PlatformECIRhoZDot;
double PlatformECIRhoXDotDot;
double *PlatformECIRhoXDotDotPtr = &PlatformECIRhoXDotDot;
double PlatformECIRhoYDotDot;
double *PlatformECIRhoYDotDotPtr = &PlatformECIRhoYDotDot;
double PlatformECIRhoZDotDot;
double *PlatformECIRhoZDotDotPtr = &PlatformECIRhoZDotDot;
double PlatformRENRhoR;
double *PlatformRENRhoRPtr = &PlatformRENRhoR;
double PlatformRENRhoE;
double *PlatformRENRhoEPtr = &PlatformRENRhoE;
double PlatformRENRhoN;
double *PlatformRENRhoNPtr = &PlatformRENRhoN;
double PlatformRENRhoRDot;
double *PlatformRENRhoRDotPtr = &PlatformRENRhoRDot;
double PlatformRENRhoEDot;
double *PlatformRENRhoEDotPtr = &PlatformRENRhoEDot;
double PlatformRENRhoNDot;
double *PlatformRENRhoNDotPtr = &PlatformRENRhoNDot;
double PlatformRENRhoRDotDot;
double *PlatformRENRhoRDotDotPtr = &PlatformRENRhoRDotDot;
double PlatformRENRhoEDotDot;
double *PlatformRENRhoEDotDotPtr = &PlatformRENRhoEDotDot;
double PlatformRENRhoNDotDot;
double *PlatformRENRhoNDotDotPtr = &PlatformRENRhoNDotDot;
double  ECItoRENMatrix11;
double *ECItoRENMatrix11Ptr = &ECItoRENMatrix11;
double  ECItoRENMatrix12;
double *ECItoRENMatrix12Ptr = &ECItoRENMatrix12;
double  ECItoRENMatrix13;
double *ECItoRENMatrix13Ptr = &ECItoRENMatrix13;
double  ECItoRENMatrix21;
```

```
    double *ECItoRENMatrix21Ptr = &ECItoRENMatrix21;
    double  ECItoRENMatrix22;
    double *ECItoRENMatrix22Ptr = &ECItoRENMatrix22;
    double  ECItoRENMatrix23;
    double *ECItoRENMatrix23Ptr = &ECItoRENMatrix23;
    double  ECItoRENMatrix31;
    double *ECItoRENMatrix31Ptr = &ECItoRENMatrix31;
    double  ECItoRENMatrix32;
    double *ECItoRENMatrix32Ptr = &ECItoRENMatrix32;
    double  ECItoRENMatrix33;
    double *ECItoRENMatrix33Ptr = &ECItoRENMatrix33;

    /**********************************************/
    /*  GET AIRCRAFT POSITION INFORMATION         */
    /**********************************************/
    ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
    strcpy(buff,HemisphereEdit->Text.c_str());
    if ((!(strcmp(buff, "N"))) || (!(strcmp(buff, "n"))))
        ABLPlatform->SetLatitudeHemisphere(0);
    else if ((!(strcmp(buff, "S"))) || (!(strcmp(buff, "s"))))
        ABLPlatform->SetLatitudeHemisphere(1);
    else
    {    ErrorList.AddError("EvaluateEphemerisForm",
                            "Lat Hemisphere must be north(N) or south(S)",
                             1);
    }
    ABLPlatform->SetLatitudeDegree(LatitudeDegreeEdit->Text.ToDouble());
    ABLPlatform->SetLatitudeMinute(LatitudeMinuteEdit->Text.ToDouble());
    ABLPlatform->SetLatitudeSecond(LatitudeSecondEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeDegree(LongitudeDegreeEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeMinute(LongitudeMinuteEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeSecond(LongitudeSecondEdit->Text.ToDouble());
    ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
    ABLPlatform->SetVelocityX(VelocityXEdit->Text.ToDouble());
    ABLPlatform->SetVelocityY(VelocityYEdit->Text.ToDouble());
    ABLPlatform->SetVelocityZ(VelocityZEdit->Text.ToDouble());

    /**********************************************/
    /*  GET GREENWICH MERIDIAN REFERENCE          */
    /**********************************************/
    ReferenceHour = ReferenceHourEdit->Text.ToDouble();
    ReferenceMinute = ReferenceMinuteEdit->Text.ToDouble();
    ReferenceSecond = ReferenceSecondEdit->Text.ToDouble();
    RefModJulianDate = RefModJulianDateEdit->Text.ToDouble();

    /**********************************************/
    /*   GET CURRENT TIME                         */
    /**********************************************/
    CalcYear = CalcYearEdit->Text.ToInt();
    CalcMonth = CalcMonthEdit->Text.ToInt();
    CalcDay = CalcDayEdit->Text.ToInt();
    CalcHour = CalcHourEdit->Text.ToInt();
    CalcMinute = CalcMinuteEdit->Text.ToInt();
    CalcSecond = CalcSecondEdit->Text.ToDouble();

/****************************************************/
/*  FIND THE CURRENT ANGLE OF THETA G AT THE        */
/*  TIME OF PROPAGATION                             */
/****************************************************/
    ThetaGInRadians = 0;
    FindThetaG(ReferenceHour,
               ReferenceMinute,
               ReferenceSecond,
```

```
              RefModJulianDate,
              CalcYear,
              CalcMonth,
              CalcDay,
              CalcHour,
              CalcMinute,
              CalcSecond,
              *ThetaPtr,
              *ErrorPtr);

/***************************************************/
/*  CONVERT THE PROPAGATION TIME TO A JULIAN DATE  */
/*  THAT CAN BE RECOGNIZED BY "TargetSatellite".   */
/***************************************************/
    JulianDate = 0.0;
    ConvertCalenderToJulian(CalcYear,
                            CalcMonth,
                            CalcDay,
                            CalcHour,
                            CalcMinute,
                            CalcSecond,
                            *JulianDatePtr,
                            *ErrorPtr);
/***************************************************/
/*  EVALUATE THE POSITION, VELOCITY AND THE        */
/*  ACCELERATION OF THE PLATFORM                   */
/***************************************************/
    TargetPlatform(*ABLPlatform,
                   *ThetaPtr,
                   JulianDate,
                   *PlatformECIRhoXPtr,
                   *PlatformECIRhoYPtr,
                   *PlatformECIRhoZPtr,
                   *PlatformECIRhoXDotPtr,
                   *PlatformECIRhoYDotPtr,
                   *PlatformECIRhoZDotPtr,
                   *PlatformECIRhoXDotDotPtr,
                   *PlatformECIRhoYDotDotPtr,
                   *PlatformECIRhoZDotDotPtr,
                   *PlatformRENRhoRPtr,
                   *PlatformRENRhoEPtr,
                   *PlatformRENRhoNPtr,
                   *PlatformRENRhoRDotPtr,
                   *PlatformRENRhoEDotPtr,
                   *PlatformRENRhoNDotPtr,
                   *PlatformRENRhoRDotDotPtr,
                   *PlatformRENRhoEDotDotPtr,
                   *PlatformRENRhoNDotDotPtr,
                   *ECItoRENMatrix11Ptr,
                   *ECItoRENMatrix12Ptr,
                   *ECItoRENMatrix13Ptr,
                   *ECItoRENMatrix21Ptr,
                   *ECItoRENMatrix22Ptr,
                   *ECItoRENMatrix23Ptr,
                   *ECItoRENMatrix31Ptr,
                   *ECItoRENMatrix32Ptr,
                   *ECItoRENMatrix33Ptr,
                   *ErrorPtr);

/***************************************************/
/*  OUTPUT THE TEST PARAMETERS WHICH MONITOR THE   */
/*  CALCULATIONS IN "TargetSatellite".             */
/***************************************************/
```

```
    XEdit->Text = String(PlatformECIRhoX);
    YEdit->Text = String(PlatformECIRhoY);
    ZEdit->Text = String(PlatformECIRhoZ);
    XDotEdit->Text = String(PlatformECIRhoXDot*3600);
    YDotEdit->Text = String(PlatformECIRhoYDot*3600);
    ZDotEdit->Text = String(PlatformECIRhoZDot*3600);
    XDotDotEdit->Text = String(PlatformECIRhoXDotDot*1000);
    YDotDotEdit->Text = String(PlatformECIRhoYDotDot*1000);
    ZDotDotEdit->Text = String(PlatformECIRhoZDotDot*1000);
    REdit->Text = String(PlatformRENRhoR);
    EEdit->Text = String(PlatformRENRhoE);
    NEdit->Text = String(PlatformRENRhoN);
    RDotEdit->Text = String(PlatformRENRhoRDot*3600);
    EDotEdit->Text = String(PlatformRENRhoEDot*3600);
    NDotEdit->Text = String(PlatformRENRhoNDot*3600);
    RDotDotEdit->Text = String(PlatformRENRhoRDotDot*1000);
    EDotDotEdit->Text = String(PlatformRENRhoEDotDot*1000);
    NDotDotEdit->Text = String(PlatformRENRhoNDotDot*1000);
    ThetaGEdit->Text = String(ThetaGInRadians * RADTODEGREES);

/******************************************/
/*  PRINT OUT ALL ERROR MESSAGES          */
/******************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }


}
```

E.8 TargetSatelliteForm.cpp

```
/************************************************************************************/
/*  MODULE NAME:     TargetSatelliteForm.cpp                                        */
/*  AUTHOR:          Captain David Vloedman                                         */
/*  DATE CREATED:    November 17, 1998                                              */
/*                                                                                  */
/*  PURPOSE:         This is the Form which can be used to test the modules        */
/*                   created in TargetSatellite.cpp.  This form                     */
/*                   takes all the inputs to evaluate a single satellite            */
/*                   ephemeris against a single airborne platform, and              */
/*                   determines the azimuth and elevation of the satellite          */
/*                   with respect to the ABL laser platform.                        */
/*                                                                                  */
/*  COMPILER:        Borland C++ Builder3 Standard version                          */
/*                   This compiler should be used to compile and link.              */
/*                                                                                  */
/************************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "TimeModules.h"
#include "LaserConstants.h"
#include "ErrorStructure.h"
#include "TargetSatelliteForm.h"
#include "Aircraft.h"
#include "Satellite.h"
#include "EvaluateEphemerisModules.h"
#include "TargetSatellite.h"
#include "TargetPlatform.h"
#include "TLEInput.h"
/**********************************/
/* C SPECIFIC LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>

/*******************************/
/*      CREATE THE FORM       */
/*******************************/
TForm1 *Form1;


__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
/*****************************************************/
/*  THIS PROCURE HANDLES THE BUTTON TO ACTUALLY RUN  */
/*  THE FINDING OF THE AZIMUTH AND ELEVATION          */
/*****************************************************/
void __fastcall TForm1::EvaluateButtonClick(TObject *Sender)
```

```
{
    ErrorStructure ErrorList;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    Aircraft* ABLPlatform;
        ABLPlatform = new Aircraft;
    Satellite* Sat;
        Sat = new Satellite;
    char    Errors[MAXERRORS][MAXMESSAGELENGTH];
    char    buff[MAXNAMELENGTH];
    double  ReferenceHour;              /****************************/
    double  ReferenceMinute;            /*  THE REFERENT ANGLE      */
    double  ReferenceSecond;            /*  OF THETA G IS KNOWN AS  */
    double  RefModJulianDate;           /*  THE REFERENCE ANGLE IN  */
    double  ThetaGInRadians;            /****************************/
    double  *ThetaPtr = &ThetaGInRadians;
    int     CalcYear;
    int     CalcMonth;
    int     CalcDay;
    int     CalcHour;
    int     CalcMinute;
    double  CalcSecond;
    int     i;
    double JulianDate;
    double *JulianDatePtr = &JulianDate;
    double SatECIRhoX;
    double *SatECIRhoXPtr = &SatECIRhoX;
    double SatECIRhoY;
    double *SatECIRhoYPtr = &SatECIRhoY;
    double SatECIRhoZ;
    double *SatECIRhoZPtr = &SatECIRhoZ;
    double SatECIRhoXDot;
    double *SatECIRhoXDotPtr = &SatECIRhoXDot;
    double SatECIRhoYDot;
    double *SatECIRhoYDotPtr = &SatECIRhoYDot;
    double SatECIRhoZDot;
    double *SatECIRhoZDotPtr = &SatECIRhoZDot;
    double SatECIRhoXDotDot;
    double *SatECIRhoXDotDotPtr = &SatECIRhoXDotDot;
    double SatECIRhoYDotDot;
    double *SatECIRhoYDotDotPtr = &SatECIRhoYDotDot;
    double SatECIRhoZDotDot;
    double *SatECIRhoZDotDotPtr = &SatECIRhoZDotDot;
    double SatRENRhoR;
    double *SatRENRhoRPtr = &SatRENRhoR;
    double SatRENRhoE;
    double *SatRENRhoEPtr = &SatRENRhoE;
    double SatRENRhoN;
    double *SatRENRhoNPtr = &SatRENRhoN;
    double SatRENRhoRDot;
    double *SatRENRhoRDotPtr = &SatRENRhoRDot;
    double SatRENRhoEDot;
    double *SatRENRhoEDotPtr = &SatRENRhoEDot;
    double SatRENRhoNDot;
    double *SatRENRhoNDotPtr = &SatRENRhoNDot;
    double SatRENRhoRDotDot;
    double *SatRENRhoRDotDotPtr = &SatRENRhoRDotDot;
    double SatRENRhoEDotDot;
    double *SatRENRhoEDotDotPtr = &SatRENRhoEDotDot;
    double SatRENRhoNDotDot;
    double *SatRENRhoNDotDotPtr = &SatRENRhoNDotDot;
    double PlatformECIRhoX;
    double *PlatformECIRhoXPtr = &PlatformECIRhoX;
    double PlatformECIRhoY;
```

```
double *PlatformECIRhoYPtr = &PlatformECIRhoY;
double PlatformECIRhoZ;
double *PlatformECIRhoZPtr = &PlatformECIRhoZ;
double PlatformECIRhoXDot;
double *PlatformECIRhoXDotPtr = &PlatformECIRhoXDot;
double PlatformECIRhoYDot;
double *PlatformECIRhoYDotPtr = &PlatformECIRhoYDot;
double PlatformECIRhoZDot;
double *PlatformECIRhoZDotPtr = &PlatformECIRhoZDot;
double PlatformECIRhoXDotDot;
double *PlatformECIRhoXDotDotPtr = &PlatformECIRhoXDotDot;
double PlatformECIRhoYDotDot;
double *PlatformECIRhoYDotDotPtr = &PlatformECIRhoYDotDot;
double PlatformECIRhoZDotDot;
double *PlatformECIRhoZDotDotPtr = &PlatformECIRhoZDotDot;
double PlatformRENRhoR;
double *PlatformRENRhoRPtr = &PlatformRENRhoR;
double PlatformRENRhoE;
double *PlatformRENRhoEPtr = &PlatformRENRhoE;
double PlatformRENRhoN;
double *PlatformRENRhoNPtr = &PlatformRENRhoN;
double PlatformRENRhoRDot;
double *PlatformRENRhoRDotPtr = &PlatformRENRhoRDot;
double PlatformRENRhoEDot;
double *PlatformRENRhoEDotPtr = &PlatformRENRhoEDot;
double PlatformRENRhoNDot;
double *PlatformRENRhoNDotPtr = &PlatformRENRhoNDot;
double PlatformRENRhoRDotDot;
double *PlatformRENRhoRDotDotPtr = &PlatformRENRhoRDotDot;
double PlatformRENRhoEDotDot;
double *PlatformRENRhoEDotDotPtr = &PlatformRENRhoEDotDot;
double PlatformRENRhoNDotDot;
double *PlatformRENRhoNDotDotPtr = &PlatformRENRhoNDotDot;
double  ECItoRENMatrix11;
double *ECItoRENMatrix11Ptr = &ECItoRENMatrix11;
double  ECItoRENMatrix12;
double *ECItoRENMatrix12Ptr = &ECItoRENMatrix12;
double  ECItoRENMatrix13;
double *ECItoRENMatrix13Ptr = &ECItoRENMatrix13;
double  ECItoRENMatrix21;
double *ECItoRENMatrix21Ptr = &ECItoRENMatrix21;
double  ECItoRENMatrix22;
double *ECItoRENMatrix22Ptr = &ECItoRENMatrix22;
double  ECItoRENMatrix23;
double *ECItoRENMatrix23Ptr = &ECItoRENMatrix23;
double  ECItoRENMatrix31;
double *ECItoRENMatrix31Ptr = &ECItoRENMatrix31;
double  ECItoRENMatrix32;
double *ECItoRENMatrix32Ptr = &ECItoRENMatrix32;
double  ECItoRENMatrix33;
double *ECItoRENMatrix33Ptr = &ECItoRENMatrix33;


/**********************************************/
/*  GET AIRCRAFT POSITION INFORMATION        */
/**********************************************/
ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
strcpy(buff,HemisphereEdit->Text.c_str());
if ((!(strcmp(buff, "N"))) || (!(strcmp(buff, "n"))))
    ABLPlatform->SetLatitudeHemisphere(0);
else if ((!(strcmp(buff, "S"))) || (!(strcmp(buff, "s"))))
    ABLPlatform->SetLatitudeHemisphere(1);
else
```

```
    {    ErrorList.AddError("EvaluateEphemerisForm",
                            "Lat Hemisphere must be north(N) or south(S)",
                             1);
    }
    ABLPlatform->SetLatitudeDegree(LatitudeDegreeEdit->Text.ToDouble());
    ABLPlatform->SetLatitudeMinute(LatitudeMinuteEdit->Text.ToDouble());
    ABLPlatform->SetLatitudeSecond(LatitudeSecondEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeDegree(LongitudeDegreeEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeMinute(LongitudeMinuteEdit->Text.ToDouble());
    ABLPlatform->SetLongitudeSecond(LongitudeSecondEdit->Text.ToDouble());
    ABLPlatform->SetAltitude(AltitudeEdit->Text.ToDouble());
    ABLPlatform->SetVelocityX(VelocityXEdit->Text.ToDouble());
    ABLPlatform->SetVelocityY(VelocityYEdit->Text.ToDouble());
    ABLPlatform->SetVelocityZ(VelocityZEdit->Text.ToDouble());

    /**********************************************/
    /*  GET GREENWICH MERIDIAN REFERENCE          */
    /**********************************************/
    ReferenceHour = ReferenceHourEdit->Text.ToDouble();
    ReferenceMinute = ReferenceMinuteEdit->Text.ToDouble();
    ReferenceSecond = ReferenceSecondEdit->Text.ToDouble();
    RefModJulianDate = RefModJulianDateEdit->Text.ToDouble();

    /**********************************************/
    /*  GET SATELLITE EPHEMERIS INFORMATION       */
    /**********************************************/
    Sat->SetSSCNumber(SSCEdit->Text.ToInt());
    strcpy(buff,ClassEdit->Text.c_str());
    Sat->SetSecurityClass(buff);
    strcpy(buff,IntIDEdit->Text.c_str());
    Sat->SetInternationalID(buff);
    Sat->SetEpochYear(EpochYearEdit->Text.ToInt());
    Sat->SetEpochDay(EpochDayEdit->Text.ToDouble());
    Sat->SetRevSquared(RevSquaredEdit->Text.ToDouble());
    Sat->SetRevCubed(RevCubedEdit->Text.ToDouble());
    Sat->SetBStarDrag(BStarEdit->Text.ToDouble());
    Sat->SetEphemerisType(EphemerisTypeEdit->Text.ToInt());
    Sat->SetElementSetNumber(ElSetEdit->Text.ToInt());
    Sat->SetInclination(InclinationEdit->Text.ToDouble());
    Sat->SetRightAscension(RightAscensionEdit->Text.ToDouble());
    Sat->SetEccentricity(EccentricityEdit->Text.ToDouble());
    Sat->SetArgumentOfPerigee(ArgumentOfPerigeeEdit->Text.ToDouble());
    Sat->SetMeanAnomaly(MeanAnomalyEdit->Text.ToDouble());
    Sat->SetMeanMotion(MeanMotionEdit->Text.ToDouble());
    Sat->SetRevAtEpoch(RevNumberEdit->Text.ToInt());

    /**********************************************/
    /*   GET CURRENT TIME                         */
    /**********************************************/
    CalcYear = CalcYearEdit->Text.ToInt();
    CalcMonth = CalcMonthEdit->Text.ToInt();
    CalcDay = CalcDayEdit->Text.ToInt();
    CalcHour = CalcHourEdit->Text.ToInt();
    CalcMinute = CalcMinuteEdit->Text.ToInt();
    CalcSecond = CalcSecondEdit->Text.ToDouble();


/****************************************************/
/*  FIND THE CURRENT ANGLE OF THETA G AT THE        */
/*  TIME OF PROPAGATION                             */
/****************************************************/
    ThetaGInRadians = 0;
    FindThetaG(ReferenceHour,
```

```
                ReferenceMinute,
                ReferenceSecond,
                RefModJulianDate,
                CalcYear,
                CalcMonth,
                CalcDay,
                CalcHour,
                CalcMinute,
                CalcSecond,
                *ThetaPtr,
                *ErrorPtr);

/****************************************************/
/*  CONVERT THE PROPAGATION TIME TO A JULIAN DATE   */
/*  THAT CAN BE RECOGNIZED BY "TargetSatellite".    */
/****************************************************/
    JulianDate = 0.0;
    ConvertCalenderToJulian(CalcYear,
                            CalcMonth,
                            CalcDay,
                            CalcHour,
                            CalcMinute,
                            CalcSecond,
                            *JulianDatePtr,
                            *ErrorPtr);
/****************************************************/
/*  EVALUATE WHETHER OR NOT THE SATELLITE IS        */
/*  CURRENTLY WITHIN VIEW OF THE PLATFORM           */
/****************************************************/
    TargetPlatform(*ABLPlatform,
                   *ThetaPtr,
                   JulianDate,
                   *PlatformECIRhoXPtr,
                   *PlatformECIRhoYPtr,
                   *PlatformECIRhoZPtr,
                   *PlatformECIRhoXDotPtr,
                   *PlatformECIRhoYDotPtr,
                   *PlatformECIRhoZDotPtr,
                   *PlatformECIRhoXDotDotPtr,
                   *PlatformECIRhoYDotDotPtr,
                   *PlatformECIRhoZDotDotPtr,
                   *PlatformRENRhoRPtr,
                   *PlatformRENRhoEPtr,
                   *PlatformRENRhoNPtr,
                   *PlatformRENRhoRDotPtr,
                   *PlatformRENRhoEDotPtr,
                   *PlatformRENRhoNDotPtr,
                   *PlatformRENRhoRDotDotPtr,
                   *PlatformRENRhoEDotDotPtr,
                   *PlatformRENRhoNDotDotPtr,
                   *ECItoRENMatrix11Ptr,
                   *ECItoRENMatrix12Ptr,
                   *ECItoRENMatrix13Ptr,
                   *ECItoRENMatrix21Ptr,
                   *ECItoRENMatrix22Ptr,
                   *ECItoRENMatrix23Ptr,
                   *ECItoRENMatrix31Ptr,
                   *ECItoRENMatrix32Ptr,
                   *ECItoRENMatrix33Ptr,
                   *ErrorPtr);

/****************************************************/
/*  EVALUATE WHETHER OR NOT THE SATELLITE IS        */
```

```
/*  CURRENTLY WITHIN VIEW OF THE PLATFORM             */
/**************************************************/
    TargetSatellite(*Sat,
                    JulianDate,
                    ECItoRENMatrix11,
                    ECItoRENMatrix12,
                    ECItoRENMatrix13,
                    ECItoRENMatrix21,
                    ECItoRENMatrix22,
                    ECItoRENMatrix23,
                    ECItoRENMatrix31,
                    ECItoRENMatrix32,
                    ECItoRENMatrix33,
                    *SatECIRhoXPtr,
                    *SatECIRhoYPtr,
                    *SatECIRhoZPtr,
                    *SatECIRhoXDotPtr,
                    *SatECIRhoYDotPtr,
                    *SatECIRhoZDotPtr,
                    *SatECIRhoXDotDotPtr,
                    *SatECIRhoYDotDotPtr,
                    *SatECIRhoZDotDotPtr,
                    *SatRENRhoRPtr,
                    *SatRENRhoEPtr,
                    *SatRENRhoNPtr,
                    *SatRENRhoRDotPtr,
                    *SatRENRhoEDotPtr,
                    *SatRENRhoNDotPtr,
                    *SatRENRhoRDotDotPtr,
                    *SatRENRhoEDotDotPtr,
                    *SatRENRhoNDotDotPtr,
                    *ErrorPtr);

/**************************************************/
/*  OUTPUT THE TEST PARAMETERS WHICH MONITOR THE    */
/*  CALCULATIONS IN "TargetSatellite".              */
/**************************************************/
    XEdit->Text = String(SatECIRhoX);
    YEdit->Text = String(SatECIRhoY);
    ZEdit->Text = String(SatECIRhoZ);
    XDotEdit->Text = String(SatECIRhoXDot);
    YDotEdit->Text = String(SatECIRhoYDot);
    ZDotEdit->Text = String(SatECIRhoZDot);
    XDotDotEdit->Text = String(SatECIRhoXDotDot*1000.0);
    YDotDotEdit->Text = String(SatECIRhoYDotDot*1000.0);
    ZDotDotEdit->Text = String(SatECIRhoZDotDot*1000.0);
    REdit->Text = String(SatRENRhoR);
    EEdit->Text = String(SatRENRhoE);
    NEdit->Text = String(SatRENRhoN);
    RDotEdit->Text = String(SatRENRhoRDot);
    EDotEdit->Text = String(SatRENRhoEDot);
    NDotEdit->Text = String(SatRENRhoNDot);
    RDotDotEdit->Text = String(SatRENRhoRDotDot*1000.0);
    EDotDotEdit->Text = String(SatRENRhoEDotDot*1000.0);
    NDotDotEdit->Text = String(SatRENRhoNDotDot*1000.0);

/******************************************/
/*  PRINT OUT ALL ERROR MESSAGES          */
/******************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
```

```
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }


}




/***************************************************/
/*  THIS EVENT HANDLER PROCEDURE HANDLES THE BUTTON*/
/*  THAT CAN LOAD A TEST CASE FROM A FILE FOR LATER*/
/*  EXECUTION                                      */
/***************************************************/
void __fastcall TForm1::FileButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
    SatStructure *SatArray = new SatStructure;

    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int i;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    char FileName[MAXNAMELENGTH] = " ";


/***************************************************/
/*  GET NAME OF FILE TO READ TEST CASE FROM        */
/***************************************************/
    strcpy(FileName,FileEdit->Text.c_str());

/***************************************************/
/*  READ ALL SATELLITES FROM THE FILE, AND USE THE */
/*  FIRST SATELLITE IN THE FILE AS THE TEST CASE   */
/***************************************************/
    ReadTLEFile(FileName,
                *SatArray,
                *ErrorPtr);

/***************************************************/
/*  NOTE THE Sat[0] IS THE FIRST SATELLITE IN THE  */
/*  FILE                                           */
/***************************************************/
    SSCEdit->Text = String(SatArray->Sat[0].GetSSCNumber());
    ClassEdit->Text = String(SatArray->Sat[0].GetSecurityClass());
    IntIDEdit->Text = String(SatArray->Sat[0].GetInternationalID());
    EpochYearEdit->Text = String(SatArray->Sat[0].GetEpochYear());
    EpochDayEdit->Text = String(double(SatArray->Sat[0].GetEpochDay()));
    RevSquaredEdit->Text = String(double(SatArray->Sat[0].GetRevSquared()));
    RevCubedEdit->Text = String(double(SatArray->Sat[0].GetRevCubed()));
    BStarEdit->Text = String(double(SatArray->Sat[0].GetBStarDrag()));
    EphemerisTypeEdit->Text = String(SatArray->Sat[0].GetEphemerisType());
    ElSetEdit->Text = String(SatArray->Sat[0].GetElementSetNumber());
    InclinationEdit->Text = String(double(SatArray->Sat[0].GetInclination()));
    RightAscensionEdit->Text              =              String(double(SatArray-
>Sat[0].GetRightAscension()));
```

```
    EccentricityEdit->Text                =                String(double(SatArray-
>Sat[0].GetEccentricity()));
    ArgumentOfPerigeeEdit->Text              =              String(double(SatArray-
>Sat[0].GetArgumentOfPerigee()));
    MeanAnomalyEdit->Text = String(double(SatArray->Sat[0].GetMeanAnomaly()));
    MeanMotionEdit->Text = String(double(SatArray->Sat[0].GetMeanMotion()));
    RevNumberEdit->Text = String(SatArray->Sat[0].GetRevAtEpoch());


/****************************************************/
/*    DISPLAY ALL ERRORS                            */
/****************************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }

}
```

E.9 TestTargetLaserForm.cpp

```
/****************************************************************************/
/*  MODULE NAME:     TargetLaserForm.cpp                                    */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    November 17, 1998                                      */
/*                                                                          */
/*  PURPOSE:         This is the Form which can be used to test the modules */
/*                   created in TargetLaser.cpp.  This form                 */
/*                   takes all the inputs to evaluate a single laser        */
/*                   trajectory in the REN frame given the azimuth (degrees */
/*                   east of north) and elevation (degrees above horizon)   */
/*                   of the laser turret.                                   */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/**********************************/
/* USER-BUILT LIBRARIES           */
/**********************************/
#include "LaserConstants.h"
#include "ErrorStructure.h"
#include "TestTargetLaserForm.h"
#include "TargetLaser.h"
/**********************************/
/* C SPECIFIC LIBRARIES           */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>

/********************************/
/*      CREATE THE FORM         */
/********************************/
TForm1 *Form1;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
/******************************************************/
/*  THIS PROCURE HANDLES THE BUTTON TO ACTUALLY RUN   */
/*  THE ROUTINE TO FIND THE LASER PARAMETERS.         */
/******************************************************/
void __fastcall TForm1::EvaluateButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    char     Errors[MAXERRORS][MAXMESSAGELENGTH];
    int      i;
    double AzimuthInDegrees;
```

```
    double AzimuthDot;
    double AzimuthDotDot;
    double ElevationInDegrees;
    double ElevationDot;
    double ElevationDotDot;
    double LaserRENRhoR;
    double *LaserRENRhoRPtr = &LaserRENRhoR;
    double LaserRENRhoE;
    double *LaserRENRhoEPtr = &LaserRENRhoE;
    double LaserRENRhoN;
    double *LaserRENRhoNPtr = &LaserRENRhoN;
    double LaserRENRhoRDot;
    double *LaserRENRhoRDotPtr = &LaserRENRhoRDot;
    double LaserRENRhoEDot;
    double *LaserRENRhoEDotPtr = &LaserRENRhoEDot;
    double LaserRENRhoNDot;
    double *LaserRENRhoNDotPtr = &LaserRENRhoNDot;
    double LaserRENRhoRDotDot;
    double *LaserRENRhoRDotDotPtr = &LaserRENRhoRDotDot;
    double LaserRENRhoEDotDot;
    double *LaserRENRhoEDotDotPtr = &LaserRENRhoEDotDot;
    double LaserRENRhoNDotDot;
    double *LaserRENRhoNDotDotPtr = &LaserRENRhoNDotDot;


/**********************************************/
/*   GET CURRENT TIME                         */
/**********************************************/
    AzimuthInDegrees = AzimuthEdit->Text.ToDouble();
    ElevationInDegrees = ElevationEdit->Text.ToDouble();
    AzimuthDot = AzimuthDotEdit->Text.ToDouble();
    ElevationDot = ElevationDotEdit->Text.ToDouble();
    AzimuthDotDot = AzimuthDotDotEdit->Text.ToDouble();
    ElevationDotDot = ElevationDotDotEdit->Text.ToDouble();


/****************************************************/
/*  EVALUATE WHETHER OR NOT THE SATELLITE IS        */
/*  CURRENTLY WITHIN VIEW OF THE PLATFORM           */
/****************************************************/
TargetLaser(AzimuthInDegrees,
            ElevationInDegrees,
            AzimuthDot,
            ElevationDot,
            AzimuthDotDot,
            ElevationDotDot,
            *LaserRENRhoRPtr,
            *LaserRENRhoEPtr,
            *LaserRENRhoNPtr,
            *LaserRENRhoRDotPtr,
            *LaserRENRhoEDotPtr,
            *LaserRENRhoNDotPtr,
            *LaserRENRhoRDotDotPtr,
            *LaserRENRhoEDotDotPtr,
            *LaserRENRhoNDotDotPtr,
            *ErrorPtr);

/****************************************************/
/*  OUTPUT THE TEST PARAMETERS WHICH MONITOR THE    */
/*  CALCULATIONS IN "TargetSatellite".          */
/****************************************************/
    REdit->Text = String(LaserRENRhoR);
```

```
    EEdit->Text = String(LaserRENRhoE);
    NEdit->Text = String(LaserRENRhoN);
    RDotEdit->Text = String(LaserRENRhoRDot);
    EDotEdit->Text = String(LaserRENRhoEDot);
    NDotEdit->Text = String(LaserRENRhoNDot);
    RDotDotEdit->Text = String(LaserRENRhoRDotDot);
    EDotDotEdit->Text = String(LaserRENRhoEDotDot);
    NDotDotEdit->Text = String(LaserRENRhoNDotDot);

/*****************************************/
/*  PRINT OUT ALL ERROR MESSAGES         */
/*****************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }


}
```

E.10 TimeTestForm.cpp

```
/*************************************************************************/
/*  MODULE NAME:     TimeTestForm.cpp                                    */
/*  AUTHOR:          Captain David Vloedman                              */
/*  DATE CREATED:    July 25, 1998                                       */
/*                                                                       */
/*  PURPOSE:         This is the Form which can be used to test the modules  */
/*                   created in TimeModules.cpp.                         */
/*                                                                       */
/*  COMPILER:        Borland C++ Builder3 Standard version               */
/*                   This compiler should be used to compile and link.   */
/*                                                                       */
/*************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/**********************************/
/* USER-BUILT LIBRARIES           */
/**********************************/
#include "TimeTestForm.h"
#include "TimeModules.h"
#include "LaserConstants.h"
#include "ErrorStructure.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
/********************************/
/*      CREATE THE FORM         */
/********************************/
TForml *Forml;



//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------


void __fastcall TForm1::CalcJulianButtonClick(TObject *Sender)
{
/*******************************************************/
/*  HANDLE THE CONVERT TO JULIAN DATE PUSH-BUTTON      */
/*  ON MOUSE CLICK                                     */
/*******************************************************/

    ErrorStructure ErrorList;
    int CYear;
    int CMonth;
    int CDay;
    int CHour;
    int CMinute;
```

```
    double CSecond;
    double JulianDate;
    double *JulianDatePtr = &JulianDate;
    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int i;
    ErrorStructure *ErrorPtr=&ErrorList;      /* A POINTER TO ERRORLIST */

    CYear = CalenderYearEdit->Text.ToInt();
    CMonth = CalenderMonthEdit->Text.ToInt();
    CDay = CalenderDayEdit->Text.ToInt();
    CHour = CalenderHourEdit->Text.ToInt();
    CMinute = CalenderMinuteEdit->Text.ToInt();
    CSecond = CalenderSecondEdit->Text.ToDouble();

/***************************************/
/* CALCULATE JULIAN DATE AND DISPLAY   */
/***************************************/
    ConvertCalenderToJulian(CYear,
                            CMonth,
                            CDay,
                            CHour,
                            CMinute,
                            CSecond,
                            *JulianDatePtr,
                            *ErrorPtr);

    JulianDateEdit->Text = String(JulianDate);


/***************************************/
/* SHOW ERRORS ON SCREEN               */
/***************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Append("THERE ARE ERRORS...");
        for (i = 0; i!=ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Append(Errors[i]);
   }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }
}


/*****************************************************/
/*  HANDLE THE CONVERT TO JULIAN DATE PUSH-BUTTON    */
/*  ON MOUSE CLICK                                   */
/*****************************************************/
void __fastcall TForm1::CalcCalenderButtonClick(TObject *Sender)
{

    ErrorStructure ErrorList;
    int CYear = 0;
    int *CYearPtr = &CYear;
    int CMonth = 0;
    int *CMonthPtr = &CMonth;
    int CDay = 0;
    int *CDayPtr = &CDay;
    int CHour = 0;
    int *CHourPtr = &CHour;
```

```
    int CMinute = 0;
    int *CMinutePtr = &CMinute;
    double CSecond = 0.0;
    double *CSecondPtr = &CSecond;
    double JulianDate = 0.0;
    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int i;
    ErrorStructure *ErrorPtr=&ErrorList;      /* A POINTER TO ERRORLIST */

    JulianDate = JulianDateEdit->Text.ToDouble();
    CYear = CalenderYearEdit->Text.ToInt();
    CMonth = CalenderMonthEdit->Text.ToInt();
    CDay = CalenderDayEdit->Text.ToInt();
    CHour = CalenderHourEdit->Text.ToInt();
    CMinute = CalenderMinuteEdit->Text.ToInt();
    CSecond = CalenderSecondEdit->Text.ToDouble();

/***************************************/
/* CALCULATE CALENDER DATE AND DISPLAY*/
/***************************************/
    ConvertJulianToCalender(*CYearPtr,
                            *CMonthPtr,
                            *CDayPtr,
                            *CHourPtr,
                            *CMinutePtr,
                            *CSecondPtr,
                            JulianDate,
                            *ErrorPtr);

    CalenderYearEdit->Text = String(CYear);
    CalenderMonthEdit->Text = String(CMonth);
    CalenderDayEdit->Text = String(CDay);
    CalenderHourEdit->Text = String(CHour);
    CalenderMinuteEdit->Text = String(CMinute);
    CalenderSecondEdit->Text = String(CSecond);


/***************************************/
/* SHOW ERRORS ON SCREEN               */
/***************************************/
    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Append("THERE ARE ERRORS...");
        for (i = 0; i!=ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Append(Errors[i]);
   }
    else
    {   ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }
}
```

E.11 TLETestForm.cpp

```
/**************************************************************************/
/*  MODULE NAME:      TLETestForm.cpp                                     */
/*  AUTHOR:           Captain David Vloedman                              */
/*  DATE CREATED:     August 18, 1998                                     */
/*                                                                        */
/*  PURPOSE:          This main form module is a test module for the routines */
/*                    which read in the Two Line Element (TLE) data file  */
/*                    that holds all of the satellite ephemeris data for the */
/*                    Predictive Avoidance algorithm that this test module */
/*                    supports.  This is only a test module, and is not used */
/*                    directly except to test the TLE input routines.  These */
/*                    routines are held mostly within the TLEInput module. */
/*                    NOTE: This is only part of the C++ code used to make */
/*                    this test code.  The rest is created automatically by */
/*                    the C++ Builder3 compiler.                          */
/*                                                                        */
/*                                                                        */
/*  COMPILER:         Borland C++ Builder3 Standard version               */
/*                    This compiler should be used to compile and link.   */
/*                                                                        */
/**************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/**********************************/
/* USER-BUILT LIBRARIES          */
/**********************************/
#include "TLETestForm.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "ErrorStructure.h"
#include "TLEInput.h"
/**********************************/
/* C STANDARD LIBRARIES          */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>

/*******************************/
/*      CREATE THE FORM       */
/*******************************/
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

/**************************************************************/
/*   THIS PROCEDURE IS TIED TO A BUTTON ON THE TLE TEST      */
/*   FORM.  WHEN CLICKED WITH A MOUSE, IT EXECUTES THE       */
/*   READING OF A FILE.                                      */
/**************************************************************/
```

```
void __fastcall TForm1::ReadTLEFileButtonClick(TObject *Sender)
{

/****************************************/
/*  VARIABLES SECTION                   */
/****************************************/
    ErrorStructure ErrorList;
    SatStructure *SatArray = new SatStructure;
    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int i;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    char buffer[MAXMESSAGELENGTH] = " ";
    char FileName[MAXNAMELENGTH] = " ";


/******************************************/
/* RETRIEVE INPUT FILE NAME FROM SCREEN */
/******************************************/
    strcpy(FileName,TLEFileEdit->Text.c_str());


/********************************************/
/* READ FILE INTO AN ARRAY OF "Satellite" */
/* CLASS OBJECTS                            */
/********************************************/
    ReadTLEFile(FileName,
                *SatArray,
                *ErrorPtr);


/******************************************/
/* PRINT HEADER TO MEMO BOX             */
/******************************************/
    TLEMemoBox->Lines->Clear();
    sprintf(buffer,"FILE NAME: %s *****************",
            FileName);
    TLEMemoBox->Lines->Add(buffer);

    sprintf(buffer,"SATELLITES IN FILE: %d *****************",
            SatArray->NumSats);
    TLEMemoBox->Lines->Add(buffer);

/******************************************/
/* LOOP THROUGH ARRAY AND PRINT ALL     */
/* DATA STORED IN ARRAY OF SATELLITE    */
/* OBJECTS TO THE SCREEN                */
/******************************************/
    for (i=0; i<SatArray->NumSats; i++)
    {   sprintf(buffer,"*************** BEGIN SATELLITE %d *****************",
                i+1);
        TLEMemoBox->Lines->Add(buffer);

        /**********************************************/
        /*  SHOW ORIGINAL TLE LINES FROM INPUT FILE  */
        /**********************************************/
        sprintf(buffer,"*********************************************************",
                i+1);
        TLEMemoBox->Lines->Add(buffer);
        strcpy(buffer,SatArray->Sat[i].GetTLELine1());
        TLEMemoBox->Lines->Add(buffer);
        strcpy(buffer,SatArray->Sat[i].GetTLELine2());
        TLEMemoBox->Lines->Add(buffer);
        sprintf(buffer,"*********************************************************",
```

```
        i+1);
TLEMemoBox->Lines->Add(buffer);


/*********************************************/
/*  NOW SHOW INFORMATION AS IT WAS STORED IN */
/*  THE ARRAY                                */
/*********************************************/
sprintf(buffer,"SSC Number:                %d",
        SatArray->Sat[i].GetSSCNumber());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"Security Level:            %s",
        SatArray->Sat[i].GetSecurityClass());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"International ID:          %s",
        SatArray->Sat[i].GetInternationalID());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"Epoch Year:                %d",
        SatArray->Sat[i].GetEpochYear());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"Epoch Day:                 %20.10Lf",
        SatArray->Sat[i].GetEpochDay());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"Revs/Day Squared:          %20.10Lf",
        SatArray->Sat[i].GetRevSquared());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"Revs/Day Cubed:          %20.10Lf",
        SatArray->Sat[i].GetRevCubed());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"BStar Drag Coefficient: %20.10Lf",
        SatArray->Sat[i].GetBStarDrag());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"Element Set Number:                %d",
        SatArray->Sat[i].GetElementSetNumber());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"EphemerisType:                %d",
        SatArray->Sat[i].GetEphemerisType());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"Inclination:               %20.10Lf",
        SatArray->Sat[i].GetInclination());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"Right Ascension:           %20.10Lf",
        SatArray->Sat[i].GetRightAscension());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"Eccentricity:              %20.10Lf",
        SatArray->Sat[i].GetEccentricity());
TLEMemoBox->Lines->Add(buffer);

sprintf(buffer,"Argument Of Perigee:     %20.10Lf",
        SatArray->Sat[i].GetArgumentOfPerigee());
TLEMemoBox->Lines->Add(buffer);
```

```
        sprintf(buffer,"Mean Anomaly:   %20.10Lf",
                SatArray->Sat[i].GetMeanAnomaly());
        TLEMemoBox->Lines->Add(buffer);

        sprintf(buffer,"Mean Motion:   %20.10Lf",
                SatArray->Sat[i].GetMeanMotion());
        TLEMemoBox->Lines->Add(buffer);

        sprintf(buffer,"Rev Number At Epoch:          %d",
                SatArray->Sat[i].GetRevAtEpoch());
        TLEMemoBox->Lines->Add(buffer);

        sprintf(buffer,"**************** END SATELLITE %d *****************",
                i+1);
        TLEMemoBox->Lines->Add(buffer);

    }
    sprintf(buffer,"END OF FILE NAME: %s *****************",
            FileName);
    TLEMemoBox->Lines->Add(buffer);



/**********************************************/
/*  PRINT ANY ERRORS TO THE ERROR MEMO BOX    */
/**********************************************/

    CreateDisplayText(ErrorList, Errors);
    if (ErrorList.TotalErrors()!=0)
    {
        TLEErrorMemoBox->Lines->Clear();
        TLEErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
        for (i = 0; i<ErrorList.TotalErrors(); i++)
            TLEErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {   TLEErrorMemoBox->Lines->Clear();
        TLEErrorMemoBox->Lines->Add("No Errors...");
    }
}
```

E.12 TestErrorStructure.cpp (Non-Graphical Interface)

```
/****************************************************************************/
/*  MODULE NAME:     TestErrorStructure.cpp                                 */
/*  AUTHOR:          Captain David Vloedman                                 */
/*  DATE CREATED:    August 15, 1998                                        */
/*                                                                          */
/*  PURPOSE:         This module is design to be a simple test module for   */
/*                   the ErrorStructure modules.  It makes calls to the     */
/*                   ErrorStructure routines and uses the error structures. */
/*                   This code is not an executable part of the PA project. */
/*                   It is only a test stub.                                */
/*                                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version                  */
/*                   This compiler should be used to compile and link.      */
/*                                                                          */
/****************************************************************************/
/**********************************/
/* C++BUILDER-SPECIFIC LIBRARIES */
/**********************************/
#pragma hdrstop
#pragma argsused
#include <condefs.h>
#include <stdio.h>
/**********************************/
/* C STANDARD LIBRARIES           */
/**********************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
/**********************************/
/* USER DEFINED LIBRARIES         */
/**********************************/
#include "ErrorStructure.h"
USEUNIT("ErrorStructure.cpp");
//---------------------------------------------------------------------------
int main()
{

/*******************************************/
/*  NOTE: ErrorStructure is defined within */
/*  ErrorStructure.h                       */
/*******************************************/
    ErrorStructure ErrorList;
    ErrorStructure *ErrorPtr=&ErrorList;
    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int count;
    int i;
    char buffer[MAXINPUTLINELENGTH];
/*******************************************/
/*        BEGIN EXECUTION                  */
/*******************************************/
    strcpy(buffer,"Test Case                        1");
    ErrorList.AddError("Main 1",buffer, 0);
    ErrorList.AddError("Main 2","Test Case                            2", 1);
    ErrorList.AddError("Main 3","Test Case                            3", 1);
    ErrorList.AddError("Main 4","Test Case 4", 1);
    ErrorList.AddError("Main 5","Test Case 5", 1);
    ErrorList.AddError("Main 6","Test Case 6", 1);
    ErrorList.AddError("Main 7","Test Case 7", 1);
```

```
    CreateDisplayText(*ErrorPtr, Errors);
    for (i = 0; i<ErrorList.TotalErrors(); i++)
        cout << Errors[i] << endl;
    i = getch();

}
```

Appendix F.
Sample Two Line Element (TLE) File Format

F.1 Sample TLE File

```
1  6909U 73081A   96095.97701855 +.00000078 +00000-0 +70284-4 0 0102
2  6909 089.7704 093.8156 0162024 105.7374 256.1702 13.6800259831943
1  6909U 73081A   96095.97701855 +.00000078 +00000-0 +70284-4 0 0102
2  6909 089.7704 093.8156 0162024 105.7374 256.1702 13.6800259831943
1  8746U 76023A   96096.15457780 -.00000097 +00000-0 +10000-3 0 0049
2  8746 015.3575 273.2760 0013056 142.8352 087.7876 01.0026721801907
1  8747U 76023B   96096.15673441 -.00000102 +00000-0 +00000-0 0 0012
2  8747 015.3521 273.3512 0023832 136.2812 095.6103 01.0027314701907
1  9478U 76101A   96093.28172880 +.00000049 +00000-0 +10000-3 0 0615
2  9478 012.9069 033.7479 0005869 099.5006 341.4463 01.0028020801458
1 10637U 78012A   96095.86463512 -.00000154 +00000-0 +10000-3 0 0244
2 10637 035.6846 084.2647 1353688 054.8200 331.5684 01.0025756801920
1 12089U 80098A   96093.11987032 -.00000271 +00000-0 +00000-0 0 0816
2 12089 006.2358 056.4726 0003308 303.2412 193.6849 01.0026574302452
1 12994U 81119A   96091.51457987 -.00000150 +00000-0 +10000-3 0 0530
2 12994 005.8775 057.5493 0005167 316.4773 157.4150 01.0027132601693
1 13083U 82017A   96088.99006407 +.00000070 +00000-0 +10000-3 0 0638
2 13083 005.8733 057.2095 0030143 271.4999 088.2606 00.9922799103227
1 13367U 82072A   96096.17389770 +.00000033 +00000-0 +17128-4 0 0894
2 13367 098.0838 149.2719 0007846 015.4314 344.7113 14.5717754372990
1 13595U 82097A   96094.59762657 -.00000001 +00000-0 +10000-3 0 0370
.
.
.
2 23741 000.0085 167.3677 0002445 201.5504 192.8057 01.0027165000118
1 23748U 95071A   96096.18479622 +.00001785 +00000-0 +34075-4 0 0242
2 23748 065.0214 133.8895 0010447 300.9161 059.0956 15.5209027201663
1 23751U 95072A   96096.13625365 -.00000044 +00000-0 +00000-0 0 0086
2 23751 098.6992 171.3332 0001121 059.0289 301.1001 14.2163508701405
1 23752U 95072B   96096.13206129 -.00000020 +00000-0 +10000-4 0 0041
2 23752 098.5532 170.2575 0004324 212.4478 147.6434 14.2488888901408
1 23754U 95073A   96095.49851374 -.00000008 +00000-0 +00000-0 0 0064
2 23754 000.0342 124.0248 0002246 262.3037 227.1814 01.0027393100088
1 23757U 95074A   96096.09776230 +.00000555 +00000-0 +17404-4 0 0063
2 23757 022.9772 189.2860 0013321 265.4654 094.4354 14.9762623401449
```

F.2 TLE Set Format

TLE files consist of a listing of two-line element sets as provided by the U.S. Space Command (USSC). TLE sets are The following table describes the format of a TLE set, which is composed of two "Cards", or lines.

Table F.1. **Format of Card 1**

Column	Description
1	Card number
2	Blank
3-7	Satellite or SSC number
8	Security classification
9	Blank
10-17	International number
18	Blank
19-20	Epoch Year
21-32	Epoch day to eight decimal places
33	Blank
34-43	N/2 - Revolutions per day squared
44	Blank
45-52	N/6 - Revolutions per day cubed
53	Blank
54-61	Bstar drag
62	Blank
63	Ephemeris
64	Blank
65-68	Element set number

***Table F.2.* Format of Card 2**

Column	Description
1	Card number
2	Blank
3-7	Satellite or SSC number
8	Blank
9-16	Inclination (degrees)
17	Blank
18-25	Right ascension of node (degrees)
26	Blank
27-33	Eccentricity (decimal point understood)
34	Blank
35-42	Argument of perigee (degrees)
43	Blank
44-51	Mean anomaly (degrees)
52	Blank
53-63	Mean motion (revolutions per day)
64	Revolution number at epoch

Bibliography

Bate, Roger R, Donald D. Mueller, Jerry E. White. Fundamentals of Astrodynamics. New York: Dover Publications, Inc., 1971.

Forden, Geoffrey E. "The Airborne Laser," The IEEE Spectrum, pg 40-49, (September 1997).

Hubbard, John. Schaum's Outline of Theory and Problems of Programming With C++. New York: McGraw Hill,1996.

Kelley, Al and Ira Pohl. C by Dissection, The Esssentials of C programming (Second Edition). Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc., 1992.

Kernighan, Brian W. and Dennis M. Ritchie. The C Programming Language. Englewood Cliffs, NJ: Prentice Hall, 1988.

Leonard, Mike (Captain, USAF). "Airborne Laser Predictive Avoidance Concept". Kirtland AFB: Airborne Laser Special Program Office, 1998.

Miano, John, Tom Cabanski, Harold How. Borland C++ Builder How-To. Corte Madera, CA: Waite Group Press (Division of Sam's Publishing), 1997.

Press, William H, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling. Numerical Recipes in C. Cambridge: Cambridge University Press, 1990.

Reisdorph, Kent. Teach Yourself Borland C++ Builder3 in 14 Days. Indianapolis, IN: Sam's Publishing, 1998.

Schildt, Herbert. C: The Complete Reference. Berkeley, CA: Osborne McGraw Hill, 1995.

U.S. Naval Observatory. The American Ephemeris and Nautical Almanac for the Year 1980. Washington: U.S. Government Printing Office, 1979.

Wiesel, William E. Spaceflight Dynamics (Second Edition). New York: The McGraw Hill Companies, Inc, 1997.

Vita

Captain David James Vloedman was born in Grand Rapids, Michigan, on September 21st, 1970. He graduated from Bedford High School, Bedford, Ohio in June of 1988. He entered undergraduate studies at the Ohio State University where he graduated with a Bachelor of Science degree in Computer and Information Science, specializing in Software Engineering. After graduating in 1993, he was commissioned through AFROTC Detachment 645 at the Ohio State University, where he received numerous awards including the ROTC Gold Metal of Valor.

His First Assignment was at Keesler AFB as a student in the Basic Communications and Officers Training course. During his training, he graduated at the top of his class, receiving the Class Excellence and Honor Graduate awards. In February of 1994, he was stationed at the United States Strategic Command, Offutt AFB, where he supervised the development of software used to examine our nation's Strategic Integrated Operations Plan. While at USSTRATCOM, he earned the Joint Service Commendation Medal for the timely development of two large scale software systems. In May 1997, he entered the Graduate Space Operations program under the School of Aeronautical and Astronautical Engineering, Air Force Institute of Technology. Upon graduation, he will be assigned as a Range Control Officer at the 45th Range Squadron, Cape Canaveral, where he will help plan and work launch operations.

Permanent Address: 7298 Chatham Court
Northfield Center, Ohio 44067
(330) 468-1982

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
March 1999

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
ANTI-BALLISTIC MIISILE LASER PREDICTIVE AVOIDANCE OF SATELLITES: THEORY AND SOFTWARE FOR REAL-TIME PROCESSING AND DECONFLICTION OF SATELLITES WITH A MOVING PLATFORM LASE

5. FUNDING NUMBERS

6. AUTHOR(S)

David J. Vloedman, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
2950 P Street
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GSO/ENY/M99-09

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Richard L. Flanders (c/o Bob Sendeck) BMC4I IPT/ABL Program
Boeing Defense and Space Group - Military Airplanes Division
P.O. Box 3707 MS 4A-32
Seattle, WA 98124-2207

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Dr William E. Wiesel
wiesel@afit.af.mil
(937) 255-6565 ext 4312

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Anti-Ballistic missile Laser (ABL) project is committed to defense against attack from enemy-launched missiles using an airborne laser platform. Wielding a laser of this scope requires that collateral satellites be protected from accidental illumination during operational use. The Predictive Avoidance algorithm is designed to predict the path of a given laser firing sequence, and perform real-time deconfliction with the ephemerides of a given set of satellites. This thesis establishes the theoretical framework of this algorithm, and develops a modular software package that can be incorporated into the fire-control system of ABL to perform real-time forecasting within given time and error budgets.

14. SUBJECT TERMS

Predictive Avoidance, Airborne Laser, Satellite Ephemeris, Software, Angle Forecasting

15. NUMBER OF PAGES
413

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL